On the Development of a Meshless Method to Study Multibody Systems Using Computational Fluid Dynamics

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy
by

David J. Kennett (MMath)

February 2013

Abstract

Multibody systems, which consist of several separate or interconnected, rigid or flexible bodies, occur frequently in problems of aerospace engineering. Such problems can be difficult to solve using conventional finite volume methods in computational fluid dynamics. This is particularly so if the bodies are required to undergo translational or rotational displacements during time-dependent simulations, which occur, for example, with cases involving store release or control surface deflection. These problems are generally limited to those when the movements are small or known a-priori. This thesis investigates the use of the meshless method to solve these difficult multibody systems using computational fluid dynamics, with the aim of performing moving-body simulations involving large scale motions, with no restrictions on the movement.

An implicit meshless scheme is developed to solve the Euler, laminar and Reynolds-Averaged Navier-Stokes equations. Spatial derivatives are approximated using a least squares method on clouds of points. The resultant system of equations is linearised and solved implicitly using approximate, analytical Jacobian matrices and a preconditioned Krylov subspace iterative method. The details of the spatial discretisation, linear solver and construction of the Jacobian matrix are discussed, and results which demonstrate the performance of the scheme are presented for steady and unsteady flows in two and three-dimensions.

The selection of the stencils over the computational domain for the meshless solver is vital for the method to be used to solve problems involving multibody systems accurately and efficiently. The computational domain is obtained using overlapping point distributions associated with each body in the system. Stencil selection is relatively straight forward if the point distributions are isotropic in nature; however, this is rarely the case in computations that solve the Navier-Stokes equations. A fully automatic method of selecting the stencils is outlined, in which the original connectivity and the concept of a resolving direction are used to help construct good quality stencils with limited user input. The methodology is described, and results, that are solutions to the Navier-Stokes equations in two-dimensions and the Euler equations in three-dimensions, are presented for various systems.

Acknowledgements

I would like to acknowledge my supervisors Professor Ken Badcock and Professor George Barakos for their assistance and support. I particularly wish to thank Professor Badcock without whom I would certainly have been lost. His encouragement and ideas to this work are very much appreciated. Thanks also to the other members of the Computational Fluid Dynamics Laboratory at the University of Liverpool, both past and present, with special thanks to Dr Sebastian Timme for his invaluable help in this project.

I would also like to thank the members of the Advanced Technology Centre at BAE Systems for welcoming me and making my industrial placement very productive and enjoyable. I am especially grateful to Nick Leppard for his help and guidance on this visit.

Last, but not least, my friends and family, near and far, shall not be forgotten for their support and patience.

This research has been supported by the Engineering and Physical Sciences Research Council (EPSRC) and BAE Systems through a CASE award.

Declaration

I confirm that the thesis is my own work, that I have not presented anyone else's work as my own and that full and appropriate acknowledgement has been given where reference has been made to the work of others.

David J. Kennett
February 2013

List of Publications

Kennett, D. J., Timme, S., Angulo, J. J., and Badcock, K. J., "Semi-Meshless Stencil Selection on Three-Dimensional Anisotropic Point Distributions with Parallel Implementation," *AIAA Paper 2013–0867*, Presented at the 51st AIAA Aerospace Sciences Meeting, Grapevine, Texas, Jan 2013.

Kennett, D. J., Timme, S., Angulo, J. J., and Badcock, K. J., "Semi-Meshless Stencil Selection for Anisotropic Point Distributions," *International Journal of Computational Fluid Dynamics* Vol. 26, Nos. 9–10, 2012, pp. 463–487

Kennett, D. J., Timme, S., Angulo, J. J., and Badcock, K. J., "An Implicit Meshless Method for Application in Computational Fluid Dynamics," *International Journal for Numerical Methods in Fluids* Vol. 71, No. 8, 2012, pp. 1007–1028

Kennett, D. J., Timme, S., Angulo, J. J., and Badcock, K. J., "An Implicit Semi-Meshless Scheme with Stencil Selection for Anisotropic Point Distributions," *AIAA Paper 2011–3234*, Presented at the 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 2011.

Table of Contents

List of Figures

List of Tables

List of Symbols

a	=	speed of sound
A	=	Jacobian matrix ($= \partial \boldsymbol{R}/\partial \boldsymbol{p}$)
b, c, d	=	meshless shape function derivatives
c	=	chord length
C_d	=	drag coefficient
C_f	=	skin friction coefficient
C_l, C_m	=	coefficients of lift and pitching moment
C_p	=	pressure coefficient
c_p, c_v	=	coefficients of specific heat
E	=	total energy
$\boldsymbol{f}^i$, $\boldsymbol{f}^v$	=	inviscid and viscous fluxes in x direction
$\boldsymbol{g}^i$, $\boldsymbol{g}^v$	=	inviscid and viscous fluxes in y direction
$\boldsymbol{h}^i$, $\boldsymbol{h}^v$	=	inviscid and viscous fluxes in z direction
I	=	identity matrix
I	=	inertia tensor
k	=	reduced frequency
k	=	turbulence kinetic energy
L	=	reference length scale
m	=	pseudo-time level
m	=	order of polynomial
M_∞	=	freestream Mach number
$\mathcal{M}$	=	number of points on which local flux depends
n	=	number of points in stencil
N	=	number of points in computational domain
N	=	shape function
$\boldsymbol{n}$	=	unit normal vector
n	=	real-time level
Pr, Pr_t	=	Prandtl number and turbulent Prandtl number
p	=	pressure
p	=	power of inverse distance in resolving vector

$\boldsymbol{p}$	=	vector of base monomials
$\boldsymbol{p}$	=	vector of primitive variables
$\boldsymbol{q}$	=	heat flux vector
$\boldsymbol{R}$	=	residual vector
R	=	gas constant
Re	=	chord Reynolds number
t	=	real-time
T	=	temperature
$\boldsymbol{t}$	=	unit tangential vector
$\boldsymbol{u}$	=	vector of Cartesian velocity components
$\boldsymbol{v}$	=	resolving vector
w	=	weighting function
$\boldsymbol{w}$	=	vector of conservative variables

Greek Symbols

α	=	freestream angle of attack
α	=	pitch angle
$\boldsymbol{\alpha}$	=	vector of reconstructed polynomial coefficients
β	=	control surface angle of deflection
γ	=	smallest distance in initial stencil
γ	=	ratio of specific heats
δ	=	boundary layer thickness
δ_{ij}	=	Kronecker delta
ϵ	=	increment or tolerance
ζ	=	mapped coordinate system
η	=	spanwise location
$\boldsymbol{\eta}$	=	basis vector
κ	=	thermal conductivity
μ, μ_t	=	dynamic viscosity and turbulent (eddy) viscosity
$\widetilde{\nu}$	=	intermediate variable of Spalart–Allmaras turbulence model
$\boldsymbol{\xi}$	=	vector of basis coefficients
ρ	=	density
$\boldsymbol{\tau}$	=	stress tensor
τ	=	pseudo-time
ϕ	=	flow variables
$\hat{\phi}$	=	meshless approximating function
ψ	=	flux limiter
Ω	=	computational domain

Subscripts

a	=	amplitude
e	=	effective value
e	=	average of flow variables within a set
h	=	halo value
i	=	interior point
i	=	star point
j	=	neighbouring point
L	=	left side of Riemann problem
o	=	orthogonal
R	=	right side of Riemann problem
t	=	turbulent
v	=	viscous model
0	=	steady state solution, mean value or initial value
∞	=	freestream value

Acronyms

6-DOF	=	six-degree of freedom
BILU	=	block incomplete lower-upper
CFD	=	computational fluid dynamics
CFL	=	Courant-Friedrichs-Lewy
DLR	=	German Aerospace Center
GCR	=	generalised conjugate residual
MDO	=	multidisciplinary optimisation
MLS	=	moving least squares
MPI	=	message passing interface
MUSCL	=	monotonic upstream–centered scheme for conservation laws
NS	=	Navier–Stokes
ONERA	=	French Aerospace Lab
OSF	=	open source fighter
PMB	=	parallel multiblock
PML	=	parallel meshless
POU	=	Partition of unity
RANS	=	Reynolds–averaged Navier–Stokes
SA	=	Spalart–Allmaras
SVD	=	singular value decomposition

Chapter 1

Introduction

The prediction of fluid flow involving moving aircraft components is a challenging problem. An example of such an application is in store release, in which the store, a weapon or countermeasure device, is released from the aircraft at high speed. Store separation analysis includes determining the trajectory during the initial stage of the release, and the identification of safe separation zones to avoid mid-air collisions between the store and the aircraft that released it. For this, important parameters such as the miss distance, which is the smallest distance between any part of the store and the aircraft during the early part of the trajectory, are needed for all of the different load configurations and flight conditions that are possible for the aircraft in question.

Originally, such parameters could only be determined with flight tests, in which the stores were dropped from an aircraft at gradually increasing speeds until the store came close to or actually hit the aircraft [1]. These tests were very dangerous for the test pilots, and expensive in terms of time, implementation and the possible loss of aircraft. For many years wind tunnels were the only alternative; though with the recent advances in computer hardware it is becoming more feasible to numerically compute the required parameters using computational fluid dynamics (CFD). Such methods were first used to provide a solution for a store in an aircraft flow field in the 1970s [2,3]. The continuing development of computer resources and numerical algorithms, has meant that CFD has matured to a state where it can work in conjunction with wind tunnel data and flight tests in aerodynamic design.

The numerical methods employed in CFD require some discretisation of the computational domain: this is most commonly achieved through the use of a mesh, also called a grid. The mesh consists of a set of non-overlapping cells, which in some way conform to the geometry of the problem, and ensure the conservation of the flow variables. The fluid flow over the geometry is simulated by transforming the governing equations of fluid dynamics into a discretised algebraic form, which is solved on the mesh to obtain numerical (as opposed to analytical) solutions. For a store release case, these equations are solved in a time accurate manner to determine the aerodynamic forces and moments

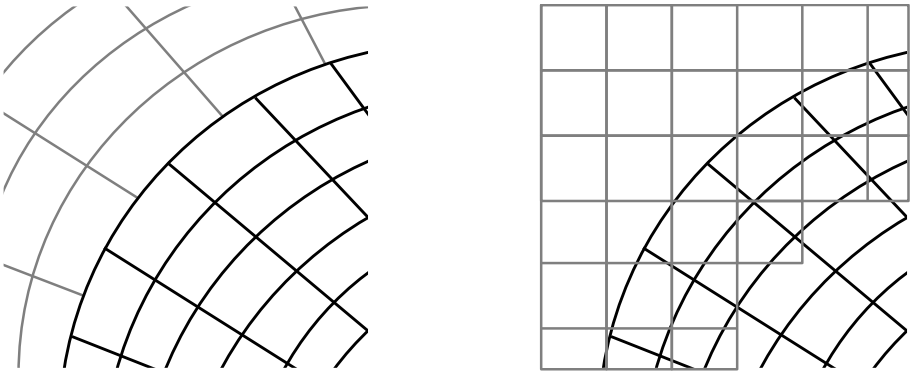
on the store. This data is then used by a six-degree of freedom (6-DOF) module to solve the rigid body equations of motion, and, thus, predict the store trajectory at each time step. The computational domain for such simulations must include, at the very least, the aircraft from which the store is released and the store itself; hence, they form what is called a multibody system. By classification multibody systems may be static; but the fluid-structure interaction between the various bodies in the store release case inevitably leads to changing geometries during the time-dependent computation. This means that there are elemental changes in the configuration over time as the bodies move relative to one another. The difficulties in handling such changes within a discretised domain to account for this movement, is one of the important issues in the simulation of this class of problem.

1.1 Moving-body problems in CFD

The most intuitive way to simulate moving-body problems is to generate a new grid at each time step, with the bodies in the new locations as predicted by the calculations made in the previous step. This means that each successive discretisation will require an interpolation of the flow solutions between the grids, which leads to inaccuracies that arise from the large number of interpolations that are needed. Also, as grid generation can be a difficult and time consuming process, remeshing the geometry after each time step is a very unattractive proposition. These disadvantages mean that researchers have looked to alternative, more elaborate techniques.

One such technique is to deform the mesh to account for the body movement. In this approach, the boundaries move according to the 6-DOF model after each time step, while the cells of the mesh expand and contract to accommodate the movement, while keeping the connectivity unchanged between time steps [4–8]. This method is relatively easy to implement into a CFD solver, and is adequate for motions that are simple. Problems arise when the bodies undergo large scale changes in their position, meaning that the required mesh deformations are excessive: then the mesh quality deteriorates and so must be reviewed during each iteration. Reference [9] presents a method in which the mesh is completely regenerated when it exhibits bad quality measures. A local remeshing procedure is used in Ref. [10], in which a hole is cut in the domain where the mesh quality has deteriorated; this region is then remeshed using an unstructured grid generator, thus, reducing where solution interpolation must occur.

Alternative methods arise by a decomposition of the domain into a set of simpler subdomains. For example, it is convenient to generate independent grids around each body separately, as subdomains, which can then be composited together in some way. If the grids meet at an interface, as in Fig.1.1(a), then they can slide past one another to simulate the relative movement between the bodies. The meshes adjacent to the sliding surface do not necessarily have to have matching nodes, or even the same



(a) Sliding mesh method (b) Overset grid method

Figure 1.1: Domain decomposition methods.

number of cell faces; hence, solutions must be passed through the interface by means of an interpolation. This method requires that the motion of the bodies be known a-priori, and is restricted to movement along the interface. Applications, for which it has been used with a great deal of success for time-dependent simulations, include turbo-machinery [11], where non-matching and rotating cell faces are used for the simulation of the flow between adjacent blade rows of engines; a study of noise generation from propeller and propeller-wing configurations [12]; and the analysis of helicopter aerodynamics [13], for which the unsteady flow over a fuselage and rotor blades is computed for a helicopter in forward flight. The restriction that the exact location of the bodies must be known after each time step means, however, that there are limited applications to the sliding mesh approach; and it cannot be used to simulate many multibody problems with unknown geometry changes, such as those that occur in store release cases.

A more flexible method is obtained if instead of the domains meeting at the interface, they are allowed to overlap, as shown in Fig.1.1(b). This means that the grids can move rigidly with their associated body components, and therefore, in theory, the bodies can move in any direction relative to one another during the computation. This grid movement can be performed without any deformation or remeshing, and communication between the grids is achieved through data interpolation. This method is called the overset grid method, though it is also commonly known as the Chimera method, and was introduced in Ref. [14] by Steger et al. in 1983. The overset grid strategy not only has the benefit of being able to simulate difficult moving-body problems, but it can also alleviate some of the difficulties in generating a full finite volume mesh, particularly for multibody systems. The technique of overlapping grids to form the computational domain means that the engineer can construct meshes around bodies, or components of bodies, independently, which is simpler and less time consuming than generating the full mesh as a whole. The Chimera method is a well-established discipline, and many

overset grid codes have been developed, such as those in Refs. [15–21]; these, and other codes, have been used to solve moving-body problems in a wide range of applications including helicopter flight [22, 23], tiltrotor flight [24], flow through a turbine [25] and store release [1, 26–32].

The use of overlapping grids means that a preprocessor stage is required to first create a computational domain on which the flow solver can operate. This stage basically consists of three steps: hole cutting, identification of interpolation cells and identification of corresponding donor cells. The hole cutting stage is necessary for regions of grids that fall into solid bodies, or other non-flow regions, when the overlap occurs: these grid regions are removed from the computational domain. Interpolation cells are needed to establish the communication of the flow solution between the grids; for each grid, these cells will lie along hole boundaries and in other regions where interpolation boundaries are identified. Donor cells are then required from the other grids to create stencils with the interpolation cells to make the inter-grid communication possible. In steady state problems these procedures only need to be performed once, as the grid geometries remain invariant during the simulation. For problems with moving-bodies the preprocessor stage has to be done after each time step, when the relative orientation of the overset grids changes.

Unfortunately, these interpolation/donor cell identification steps can be very difficult. Ideally the donor and recipient cells will be of the same size in the overlap region; if this is not the case, such as when one grid is highly refined, this mismatch can cause serious errors in the interpolation. This is because cells with lower resolution do not capture the flow features that form in the areas of high resolution; and, consequently, the magnitude of numerical error increases with the gradients of the flow variables being communicated [33]. This difficulty is often dealt with by the inclusion of another grid as a transition between the mismatching grids. This will increase the computational costs, as a three-fold grid overlap in three-dimensions can be expensive, and additional complexities in the above steps may arise. Also, the possibility of this happening at multiple geometric orientations during an unsteady, moving-body simulatation means that this fix is not ideal. There is also the problem of orphan points, which are interpolation cells for which no valid interpolation is possible; this can happen, for example, if there is insufficient overlap of the grids. Accuracy can also be compromised if an instance occurs such that the donor elements in a grid are also interpolation locations. This communication of flow data between grids is more problematic for viscous cases. These difficulties mean that despite the successful applications of the Chimera method, there are still limitations on its automatic use in moving-body problems when the bodies can move anywhere in the domain.

The above techniques all use the finite volume method, which uses cells to enforce the conservation of mass, momentum and energy of the fluid; as such, there is a clearly defined connectivity of nodes, edges and faces. Therefore, in a moving-body simulation,

if a cell comes into contact with another cell that has been used to discretise the flow around another body, then how to deal with the subsequent form of the cells, and the fluid inside each, is a major concern. This problem of the collision of such rigid structures invites the idea of breaking the restrictions that the cells impose. If only the nodes were used, then there should be no problem in moving bodies relative to one another: this is the basic idea behind the meshless method.

Meshless methods are characterised by the domain Ω requiring only a distribution of points for the solution of the equations. The flow variables ϕ are each represented by an approximating function $\hat{\phi}$, and are calculated by the use of local clouds, or stencils. In the finite volume method, the surrounding cells contribute to the stencil of each cell; in the meshless technique, each point i has a local subdomain of neighbouring points Ω_i that forms the stencil, as shown in Fig. 1.2. As cells are not used at all, then some of the difficulties associated with grid generation can be avoided. In a similar manner to the Chimera method, one can generate a distribution of points around each body separately, which adequately resolves the geometric features, and the union of the point distributions can then form the full computational domain on which the solver directly operates. Any incremental changes in the geometry due to the bodies moving during a simulation will not require a complete regeneration of the points: only the local stencils must change if the points within have moved. This is also beneficial in the design stage of aerospace vehicles: if only one component of the geometry needs to be changed, then only a regeneration of the points where the changes have occured is necessary.

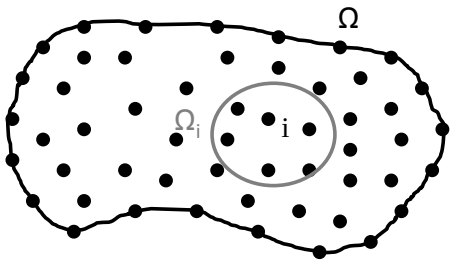


Figure 1.2: Form of a meshless domain.

As the separate distribution of points around each body is treated as an entity by the flow solver, then, unlike with the Chimera method, there is no interpolation process involved. This means that the rigorous and costly identification of donor and interpolation cells is not required, as the equations are solved directly in the regions where the point distributions overlap. It also means that there is no restriction on the resolution of the various point distributions that overlap; and the bodies are, therefore, free to move in any position relative to one another. This robustness is highly desirable with the large scale movements involved in some moving-body problems. The construction of the stencils on the set of points may, however, be equally as difficult as the identification of the interpolation stencils for the Chimera method. Research into algorithms for stencil selection are very important to determine if the method has the potential to be a practical alternative to the Chimera method for problems involving multibody systems; some of the issues regarding this process are outlined later in Section 1.4.

1.2 Meshless methods

The Smoothed Particle Hydrodynamics (SPH) method [34, 35] is generally considered to be the first meshless method; and was introduced in 1977 to model astrophysical phenomena. Despite its initial success, it was only after the 1990s that SPH was applied to a wider range of problems, such as impact, magnetohydrodynamics, heat conduction and computational mechanics [36, 37]. In the SPH method an integral representation of a function, called a kernel approximation, is used to solve the governing partial differential equations using a Lagrangian approach. The Reproducing Kernel Particle Method [38] was proposed to correct the lack of consistency in SPH, by adding a correction function to the base kernel approximation, and so improving the accuracy.

Another class of meshless method, which was proposed later and has its origin in data fitting, can be obtained if one uses a moving least squares (MLS) interpolation to determine the approximation functions used for the governing equations. The Diffuse Element Method [39] was the first to use such a procedure; it was later refined and modified in the Element Free Galerkin (EFG) method [40]. EFG has become one of the most popular of the meshless methods, and has been applied to a wide range of problems, such as fracture and crack propogation, wave propogation, acoustics and fluid flow [41–44].

As pointed out in Ref. [45], each of these methods share many common features, and, in most cases, MLS methods are identical to kernel methods. Any kernel method in which the parent kernel is identical to the weight function of a MLS approximation, and is rendered consistent by the same basis, is identical. In other words, a discrete kernel approximation which is consistent must be identical to the related MLS approximation. Underlying the two methods is the concept of the partition of unity (POU), which provides a rational method for constructing localised approximations to global functions with a greater degree of flexibilty. The POU concept is used in the hp-cloud method [46] in conjunction with a MLS interpolation.

For all of these methods, the approximation function $\hat{\phi}$ within each subdomain Ω_i, for the point i, can be expressed in the form

$$\hat{\phi}_i(\mathbf{x}) = \sum_j \mathbf{N}_j(\mathbf{x})\phi_j \tag{1.1}$$

where the sum is taken over each of the points j in the subdomain, and $\mathbf{N}_j$ is the local shape function. This form is also used in many finite element, including finite volume, discretisations. The major difference, however, is that the local shape functions are not reciprocal: this means that the shape function between a point and its neighbour within one stencil, is not the same as when the star is a point within the stencil of the neighbouring point. Reciprocal shape functions mean that all interior fluxes cancel within the domain; hence, the sum of all flux contributions reduces to the flux through

the domain boundaries. As a result, the meshless scheme cannot guarantee the conservation of the flow variables in the same way that conventional finite volume methods can. The non-reciprocal shape functions make meshless calculations much slower than their mesh based counterparts; and, more importantly, the non-conservation may lead to reduced accuracy. How much the lack of strict conservation will affect the solution, especially for turbulent flows, is an area of ongoing research; though the issue has lead to doubts about the use of meshless methods in the scientific community.

As the various meshless methods are theoretically similar, a more useful way of classifying them is by their implementation. Generally, meshless methods can be classed as either Galerkin type methods or point collocation methods. Galerkin type methods solve the weak form of the governing equations; formulations based on this form can produce a stable set of algebraic equations, though integration procedures are required since the weak form satisfies the global integral form of the governing equations. This integration is often done by the use of a background grid, which is used to create a structure to define the quadrature points; consequently, such methods are not truly meshless. In practical terms though they can still be called meshless, as the grid required is very simple, and does not need to be compatible with the points in the domain. The Meshless Local Petrov-Galerkin method [47], however, is based on the weak form, though it has a local nature in which the integral in the weak form is satisfied over a local domain. As such, a global background integration is not required: only much simpler local integrals. This scheme has been used to solve the incompressible Navier-Stokes equations in Ref. [48]. Point collocation methods, on the other hand, solve the strong form of the governing equations on the set of points. Although obtaining the exact solution for a strong form system is often more difficult and less stable, point collocation methods are less complicated to implement, and are much less costly as no background grid integration is required. It is for their speed and flexibility that point collocation methods are generally preferred in CFD. The clouds of points are used to solve the equations by first discretising the derivatives of the partial differential equation. This is done using an equivalent form of Eq. (1.1), which can be written

$$\frac{\partial \hat{\phi}_i}{\partial x} = \sum_{j \neq i} b_{(i)j} (\phi_j - \phi_i) \tag{1.2}$$

where $b_{(i)j}$ is another local shape function for the subdomain i, associated with the derivative of $\hat{\phi}_i$ with respect to x; in this form, these shape functions must obey the property

$$\sum_j b_{(i)j} = 0 \tag{1.3}$$

which is the equivalent of the partition of unity; and is called the partition of nullity for shape function derivatives. The local shape functions of Eq. (1.2) are often obtained by

Method	System	Approximation
Smoothed Particle Hydrodynamics	Strong	kernel
Reproducing Kernel Particle Method	Strong or weak	kernel
Diffuse Element Method	Weak	MLS
Element Free Galerkin	Weak	MLS
Meshless Local Petrov-Galerkin	Weak	MLS
Finite point method	Strong	MLS
hp-cloud	Weak	POU, MLS

Table 1.1: Summary of classification of meshless methods by system as strong or weak, and approximation by kernel, moving least squares (MLS), or partition of unity (POU).

performing a local least squares approximation. Least squares methods already see wide use in more traditional CFD methods, often as a means of reconstructing higher order variables in finite volume schemes [49]; meshless collocation methods are effectively an extension of this, in that we use least squares to directly compute the flux derivatives. The discretisation of the unknown function and its derivatives are defined only by the position of the points, which means that domain overlap and multibody configurations can be accommodated [46].

The first use of point collocation meshless methods to solve problems in CFD was by Batina [50], in which the Euler and Navier-Stokes equations were solved using an unweighted least squares discretisation. This method was then used in Ref. [51] for simple three-dimensional problems. This work can be seen as a precursor to the finite point method developed by Oñate et al., which was used for convection-diffusion problems and compressible fluid flow [52, 53]. Improvements to the scheme include a suitable mapping for increased stability [54], and an iterative QR decomposition method for more robust shape function computation [55].

Most meshless methods used for CFD in the literature (including this work) use the finite point method or some variant. An example of such a variant is in the use of a Taylor series representation of the approximating function, as opposed to a polynomial representation. A Taylor series representation has been used in the Least-Squares Kinetic Upwind Method (LSKUM) [56, 57], which uses a meshless discretisation of the Boltzmann equation, leading to an upwind scheme at the Euler level after taking moments. The stencils used are split stencils; and are selected according to the coordinate positions of the neighbouring points. The use of such stencils gives split flux derivatives and the upwind character to LSKUM. Praveen developed the Kinetic Meshless Method [58], which differs from LSKUM primarily in that a single stencil is used at each point; the upwinding is then introduced with the help of a modified least squares approximation, called the dual least squares approximation. Katz and Jameson [59] have developed a meshless scheme in two-dimensions that is used within an edge based

framework using grid connectivities. A multicloud algorithm, which enables simple and automatic coarsening procedures to accelerate convergence for explicit schemes, was also presented in Ref. [60]. A comparison is made between methods based on Taylor expansion, polynomial basis and radial basis function schemes in Ref. [61]. It is shown that for transonic flows with shocks, both the polynomial and Taylor series representations performed well; however, there was a slight improvement in the lift and drag coefficients when a polynomial representation is used.

A summary of the meshless methods described briefly in this section is given in Table 1.1; for a more detailed overview of the methods see Refs. [45, 62, 63].

1.3 Features of meshless methods in CFD

Another potential benefit of meshless methods is in the reduced effort required on behalf of the user to obtain CFD results. The lack of a mesh means that there is no predefined connectivity; this provides flexibility in adding or deleting points from a domain during a computation for the purpose of automatic, adaptive refinement. Such techniques are designed to capture physical features, for example shock waves, with high resolution. In finite volume methods, this involves an often complicated procedure of splitting the cells in the areas of high gradient: the so called h-adaptivity. As meshless methods require only a set of arbitrarily distributed points, an operation placing additional points in these high gradient areas is a much simpler task. These regions are often identified by the use of some sort of error estimate at each point, which checks the resolution of the solution. In a similar manner, one can easily take points away from regions where the flow is constant to save computational costs. The use of adaptive procedures means that reduced effort is required in generating the points; and accurate computation of the smaller scales of the flow field can be made, especially when we do not have a-priori information concerning the solution.

Some examples in the literature include Ref. [64], which uses an error indicator to construct a list of points to be refined; additional points are then inserted to surround each of the points in the list. This is used in conjunction with an increased order of polynomial in the approximating function, to form the full hp-adaptivity technique. Error estimates that use the difference in gradients at points and their neighbours are also seen in Refs. [65] and [66]. The former adds the additional points using a Voronoi/Delaunay technique; the latter adds points at the midpoint of the edge to be refined. Instead of inserting and deleting points from the domain, one can alternatively move the points around so that the errors in the estimators are minimised. This will keep the computational costs constant between the refinement stages, as the points gradually approach the vicinity of the important flow features. This movement is generated using a spring analogy between the points in Ref. [67], while a reference radius is adjusted in Ref. [68].

The flexibility of meshless methods means that they can be used alongside other CFD techniques. One such example is in the enforcement of boundary conditions for embedded boundary systems: these systems arise in the use of non-conforming grids. The most common type of grid used is the Cartesian mesh due to ease of generation, low storage requirements and low operation counts per cell compared to body-fitted grids; also, the lack of skewness and distortion of the cells often leads to improved convergence properties. The use of such grids means that the cells will often extend through the surface of solid components. In finite volume boundary embedded schemes, this problem can be overcome by the use of cut-cell methods. These methods can be very complicated, and lead to extremely small cells near the boundary, which may not be ideal. Instead, one can use the meshless method to implement the boundary conditions, which is much simpler. Reference [69] uses such a scheme to impose inviscid, slip boundary conditions on aerofoils: a simple finite difference numerical method is used on the Cartesian nodes, and the surface points have meshless stencils formed from the points making up the intersected cells. Additional points can be added in the sparse regions between the boundaries and Cartesian cells, formed as a result of the intersection with the body geometry, to improve the stencil quality and resolution in this region. Reference [70] presents a similar method to solve the Euler equations, but a finite volume method is used on the Cartesian mesh, and a simpler meshless approach is employed in discretising the surface.

In the schemes outlined above, only a small fraction of the computational domain is meshless. As meshless methods are typically more computationally expensive than finite volume methods, these techniques exploit the advantages of the meshless method, but keep the costs low. To perform viscous calculations, however, it is not practical to use solely unit aspect ratio Cartesian cells throughout the domain, as can be used for inviscid calculations. For this reason, the boundary embedded scheme has been extended to other methods, using more of a hybrid of meshless, and finite difference or finite volume techniques to solve the Navier-Stokes equations. The meshless clouds are preferred to resolve the viscous flow features, so there are more meshless layers of points than with the boundary embedded schemes. The Upwind Least Squares based Finite Difference solver [71, 72] is one of the few meshless solvers to solve the Reynolds-Averaged Navier-Stokes (RANS) based turbulent flow equations; this is done in Ref. [73] using a hybrid of body-fitted cells, Cartesian cells and points in which the meshless computations take place. The body-fitted cells have very high aspect ratio to capture the turbulent effects, so are only a few layers thick, the Cartesian cells are generated in the inviscid region, and the meshless points are located in the interface. A preprocessor stage is needed to cut holes in the domain where grids overlap other grids and solid bodies; this preprocessor stage is also to classify each of the points, as there are multiple solution methods involved. The points obtained from the body-fitted grid use a Green-Gauss procedure to find the gradients, to overcome the difficulty of

computing least squares derivatives on highly stretched point distributions; the points from the Cartesian grid use a finite difference methodology to compute the derivatives inexpensively; and a traditional, least squares meshless procedure is used on the points that fall between the body-fitted and Cartesian point distributions.

Hybrid methods have also been used to solve three-dimensional flow problems. A Singular Value Decomposition Generalised Finite Difference scheme [74] solves viscous problems using meshless clouds generated from the boundary surfaces in successive layers. The support nodes are selected simply by using the closest points contained within the cells of the background Cartesian mesh, thus, the procedure can operate across the different grid schemes. This method has been used further in Ref. [75] for the problem of bodies undergoing large boundary movement, namely free falling and rotating spheres: the Cartesian nodes remain stationary, while the meshless nodes move with the motion of the bodies with which they are associated. A similar hybrid method was used in Ref. [76]; in this paper, blocks of Cartesian grids were generated, inside which a box is cut to contain the geometry of the problem. The box is then filled with meshless points, generated from an unstructured grid generator; and transition points, which form the vertices of the Cartesian cells next to the boundary of the meshless region, provide the communication between the two regions. This solver format was used to study a time-dependent, inviscid store release case.

1.4 Stencil selection

Some of the solvers and methods described above may, on first thought, seem contradictory, since point distributions obtained from meshes are used to perform meshless computations. One can argue, however, that meshes have strict requirements regarding the domain decomposition, which is lost when a solver uses only the point locations: the concept of the cell and volume, the fundamental parts of a mesh, are ignored by a meshless solver (or the meshless part of a hybrid solver). The use of generated grids for meshless computations means that there will be an adequate distribution of points to resolve the geometric features that make up the problem. It also means that any type of grid can be used by the meshless solver, irrespective of its topology; and the quality of the grids is not as strict as for the finite volume solvers, so requirements such as positive volume, orthogonality, smoothness and skewness are not an issue. As mentioned, some of the difficulties associated with generation are alleviated further when one generates points for a meshless computation around bodies, or components of bodies, individually; the full point distribution is then obtained from combining these point distributions. Nevertheless, some effort must still be expended in obtaining the points; and, unless a generic, fully automatic point generation strategy is imposed, it is not possible to *completely* harness the advantages of using a meshless solver.

As well as obtaining the points for meshless methods, how the points form a connectivity is also a concern. The stencils for each point used by the meshless solver take the form of a set of neighbouring points on which the least squares approximations can take place. The full connectivity obtained in the meshing process has been used in many papers concerning meshless solvers; the most common way is to simply use the nodes from neighbouring cells in the grid connectivity to form the stencils. The test cases for which they are used are generally steady state problems for the purpose of solver validation; the points are therefore stationary, and so there is no reason to change the connectivity. Unsteady calculations with fixed grid connectivities are performed in Ref. [77], in which an aerofoil is allowed to pitch and plunge within a fixed domain. The points change position to accommodate this movement, though the stencil connectivities do not change due to the mapping used during the point movement. The motion of the points during the simulation are very similar to those in the finite volume mesh deformation method described earlier.

The main reason for the development of meshless methods, however, is for their greater flexibility and freedom in the domain decomposition. This means that the points that make up the domain would not start with a full connectivity; and so one would need a method to establish the stencils for each point in a preprocessor stage. Therefore, the problem of stencil selection is unavoidable if meshless methods are to satisfy their purpose for practical CFD calculations, otherwise the more established finite volume methods could be used instead. As the choice of points in a stencil can seriously affect accuracy and cost, this process is crucial if the method is to be competitive with finite volume methods. Despite its importance, the problem of appropriate stencil selection is, unfortunately, one that has not been addressed a great deal in the literature [78]. The construction of the local clouds is not trivial, and, although some papers have been published on this subject, which are to be discussed, there does not appear to be a fully robust, efficient method for choosing the best stencils from any point distribution.

The points chosen for each stencil must be adequate for the least squares approximations to take place; so there must be a minimum number of points that make up the clouds to result in an overdetermined linear system. The points must not be arranged to give a singular least squares matrix: so they cannot be collinear in two-dimensions, or coplanar in three-dimensions. One must also choose the points so that the functions derived approximate the fluid flow accurately; for example, a boundary layer cannot be resolved accurately if a point that lies inside the highly viscous region uses flow values from points deep within the inviscid region of the flow in its stencil. Making sure that this is not the case for random point distributions, particularly if they are not well spread or isotropic, is very difficult.

Other issues include cost and automation. For any CFD problem in which the bodies do not move, the stencil selection only has to be done once; as a result, there is some scope for user input, and the choice of using a meshless method may be justified

if the time taken to create the connectivity is less than the time needed to completely mesh the problem. For use with time-dependent, moving-body problems, in which the movement is not limited or known beforehand, the stencils will have to be selected after each time step when the points move relative to one another. Therefore, there can be no user input for such cases; and as many stencil selection processes must take place within the course of a computation, the selection must be done quickly and automatically.

A simple quadrant procedure in two-dimensions is described in Ref. [46], in which the closest points in each quadrant around a given point are chosen for the stencil. This is an improvement on a very simple nearest neighbour approach; and ensures that there are points in all directions around the point at which the flow variables are calculated, which means that the resultant stencils will be well-conditioned. A Delaunay triangulation of local candidate points is presented in Ref. [79], in which the points that make up the triangles around the point (the first layer) are retained for the stencil. This method is effectively a meshing procedure, with the stencils looking similar to those of unstructured grids. Reference [80] uses a Gauss-Jordan pivoting method on a set of points to select the best points so that the shape functions satisfy the Kronecker-delta property

$$b_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

This means that the consistency conditions and the partition of unity will be satisfied; though it means that the number of points in the stencil will always be equal to the number of unknown coefficients in the function to be approximated. Stencils are selected with the property of positivity for the Laplace operator in Ref. [81]; positive stencils obey the POU of Eq. (1.3), and have a central local shape function coefficient $b_{(i)i}$ that is negative, while the neighbouring coefficients are positive. This means that the values of $b_{(i)j}$ in Eq. (1.2) are all positive; and so a local minimum cannot decrease, and a local maximum cannot increase. These stencils lead to an M-matrix structure, which is beneficial for linear solvers; however, there are necessary assumptions on the point distributions that are made to ensure that positive stencils always exist. Geometrically, this condition is a cone criterion, in which points must be contained within a cone with a defined opening angle in whichever direction the cone points. If this is not the case, then the points are defined as not distributed nicely, and positive stencils are not guaranteed. A black box rule is given in Ref. [82] to select stencils of five points, which are the nearest neighbours that satisfy a regularity restriction in the selecting area. To do this, there are conditions which include restrictions on the collinearity of the points, and whether these points lie on a defined quadratic curve.

The stencil selection methods described above can be classed as purely meshless methods, as they attempt to select stencils from points which have no initial connec-

tivity. Although this approach is ideal, the difficulties associated have lead to methods that use some limited or initial connectivity as a guide to produce stencils: these approaches can be classed as semi-meshless. The hybrid schemes described in Section 1.3 use the body-fitted and Cartesian mesh connectivities to form the meshless clouds in the interface, so fall into this category. Similarly, a method in which clouds are formed using the cell structures of grids is given in Ref. [83]. The grids are overlapped fully like in the Chimera method, and intersecting cells from each mesh form the stencils that link them. The original connectivities are used throughout the rest of the domain for the meshless solver. A set of lines eminating from boundary walls, defined as a semi-mesh, is used in Ref. [84] as an aid in the stencil selection. These lines are used to measure the orthongonality of the points; and a merit function is used that balances the point distances and this orthogonality, to rank the neighbouring candidate points as the best to form the stencil.

As noted in Ref. [85], the type of stencils selected should depend on the equations that are to be solved; and so a stencil selection algorithm should be developed, for a specific set of equations, with this in mind. The reference, for example, uses the Laplace operator, which is isotropic, so the stencils should mimic this property; however, such stencils could cause instabilities if a hyperbolic class of equation is to be solved, and other stencil methods are preferable. The Navier-Stokes equations are not isotropic; and so the use of isotropic point distributions (or those with near unit aspect ratio) are not practical to solve them. Instead, the point distributions are anisotropic in regions near the boundary wall or in the wake, to capture high gradient flows efficiently; consequently, a stencil selection algorithm that is to be used to solve the Navier-Stokes equations should be developed with point distributions of this sort in mind. It means that classic nearest neighbour algorithms (or variants of) are generally not sufficient for the task of selection over the entire domain. This poses difficulties regarding the efficient selection of accurate, well-conditioned stencils on such point distributions, which this thesis proposes to address.

1.5 Aim of work and outline of thesis

The aim of this work is the development of a CFD technique for the simulation of moving-body flows. The meshless method is used to solve the flow equations globally on a domain consisting of overlapping point distributions, which have been generated around each of the bodies independently. These point distributions are used to solve the Navier-Stokes equations, so are anisotropic in form. Using overlapping point distributions, which are obtained from structured grids, means that there should be a sufficient number of points around the bodies to resolve the flow; and moving-body cases can be simulated by moving the input domains belonging to the bodies individually. The stencils are constructed for the solver at each time step using a method that uses the

initial connectivity of the individual point distributions as a guide, so the method can be classified as semi-meshless; though the method is not a hybrid method, and putting precedence on a single grid, or regions of several grids is not performed. For this work, we also avoid adding, moving or removing points from the domain to achieve the stencils. The majority of the work presented is with regards to the development of a new code at the University of Liverpool that has been written partly for the purpose of solving multibody problems, called Parallel MeshLess (PML) [86–91]. This thesis is divided into two parts, for each of the main functions of the code: Part I consists of Chapters 2, 3 and 4, and concerns the development of the meshless flow solver; Part II consists of Chapters 5 and 6, and concerns the stencil selection method and application to multibody systems.

Chapter 2 outlines the Navier-Stokes equations, which are the governing equations of fluid dynamics that are solved in this work. The spatial discretisation of these equations using the meshless method is described: the method closely resembles the finite point scheme mentioned earlier, in that the meshless approximation takes the form of a polynomial function, which is found using a weighted least squares method. The calculation of the shape functions that are used to find the flux derivatives, and the method of enforcement of the required boundary conditions is described.

Chapter 3 presents the temporal discretisation and the method of integration to solve the differential equations provided by the meshless discretisation. The equations are solved implicitly using an iterative, preconditioned Krylov subspace method with approximate, analytical Jacobian matrices. The parallelisation of the solver, so that calculations can run on several computers, is also described.

Chapter 4 concerns validation to build confidence in the PML flow solver. Standard inviscid, laminar and turbulent test cases for various steady and unsteady flows are computed in two and three-dimensions; and the results are compared with experimental data and other finite volume CFD solvers.

Chapter 5 presents the stencil selection method that is used to create the connectivity between the points that is used by the meshless flow solver for multibody systems. The algorithm is described for two and three-dimensional cases; and is designed to select robust stencils from any point distribution, regardless of their relative positions and form. The novel concept of the resolving vector for accurate resolution of the flow features is used for this purpose, and is applied to various steady state problems.

Chapter 6 uses the stencil selection method of Chapter 5, and is applied to multibody systems in which some of the bodies are moving in time-accurate computations. The additional complexities of simulating flows with moving-bodies, such as point velocities and solution interpolation, are described; and the method is tested finally by simulating a store release from a fighter aircraft in three-dimensions, which was the target application of this work.

Part I

Meshless Flow Solver

Chapter 2

Spatial Discretisation

2.1 Navier-Stokes equations

The governing equations of fluid dynamics considered are the Navier-Stokes equations. These equations are derived from the principle of conservation of mass (continuity), momentum (Newton's second law) and energy (the first law of thermodynamics) of the fluid. For a Newtonian, isotropic fluid with no heat sources, body forces or volumetric heating, the Navier-Stokes equations can be written, for density ρ, components of Cartesian velocity u_i and total energy E, in differential, conservative form as

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} &= 0 \\ \frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} + \frac{\partial p}{\partial x_i} &= \frac{\partial \tau_{ij}}{\partial x_j} \\ \frac{\partial (\rho E)}{\partial t} + \frac{\partial}{\partial x_j}[u_j(\rho E + p)] &= \frac{\partial}{\partial x_j}(u_i \tau_{ij} - q_j) \end{aligned} \tag{2.1}$$

where t is the time, p is the pressure, τ_{ij} is the viscous stress tensor and q_i is the heat flux vector. Respecting Stokes' hypothesis for an isotropic, Newtonian fluid, the viscous stress tensor can be written in component form as

$$\tau_{ij} = \mu \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \frac{\partial u_k}{\partial x_k} \right]$$

where μ is the coefficient of viscosity (dynamic viscosity) and δ_{ij} is the Kronecker delta symbol. The heat fluxes can be obtained using Fourier's law of thermal conduction, which states that

$$q_i = -\kappa \frac{\partial T}{\partial x_i}$$

where κ is the thermal conductivity and T is the temperature. To close this system of equations we can use the equation of state for a perfect gas, which assumes that there

are no intermolecular forces

$$p = \rho RT$$

where R is the universal gas constant. If the gas is assumed to be a calorically perfect gas, then the following relations are valid

$$C_v T = \left(E - \frac{1}{2} u_i u_i \right) \qquad \gamma = \frac{C_p}{C_v} \qquad C_v = \frac{R}{\gamma - 1} \qquad C_p = \frac{\gamma R}{\gamma - 1}$$

where C_v is the specific heat at constant volume, C_p is the specific heat at constant pressure and γ is the ratio of specific heats, which is set to be 1.4 for air.

These equations can be written with the variables given in dimensionless form, obtained by applying reference freestream values of the variables (denoted with the subscript ∞), as well as a characteristic length scale of the problem. The dimensionless parameters are the freestream Mach number M_∞, the Reynolds number Re and the Prandtl number Pr. These parameters are obtained from

$$M_\infty = \frac{u_\infty}{\sqrt{\gamma R T_\infty}} \qquad Re = \frac{\rho_\infty u_\infty L}{\mu_\infty} \qquad Pr = \frac{C_p \mu_\infty}{\kappa_\infty}$$

where L is a reference length scale and κ is the thermal conductivity; the Prandtl number is 0.72 for air.

The coefficients of viscosity and thermal conductivity can be related to the thermodynamic variables using kinetic theory. For example, Sutherland's formula gives the viscosity, from which the thermal conductivity can then be obtained

$$\mu = \mu_\infty \left(\frac{T}{T_\infty} \right)^{\frac{3}{2}} \frac{T_\infty + 110.4}{T + 110.4} \qquad \kappa = \frac{C_p \mu}{Pr}$$

This system of equations is now mathematically closed, and can be solved provided that appropriate boundary conditions are supplied. An adiabatic, no-slip boundary condition is applied at solid surfaces. This means that the velocity is zero on the solid wall, and there is no heat flux through the normal of the wall. As the flows of concern are external flows, the outer boundaries are set to uniform freestream values.

2.1.1 Reynolds-averaged Navier-Stokes equations

When the Reynolds number is sufficiently low, the fluid flows with no disruption between layers: this type of flow is called laminar. As the Reynolds number is increased, the flow becomes unstable and transitions to a turbulent flow. Turbulence has a range of scales that are linked to structures in the flow called turbulent eddies. Resolving all of these turbulent scales, even for the simplest turbulent flow, exceeds available computing power by orders of magnitude [92]; therefore, appropriate simplifications must be considered.

In a turbulent flow, the flow variables fluctuate around mean values; as such, one can present the instant flow variables as the sum of a mean flow value and a fluctuating part

$$\phi = \overline{\phi} + \phi'$$

An averaging process is introduced in order to obtain the laws of motion for the mean, turbulent quantities. Methods of averaging include time, spatial and ensemble averaging. The most common method is time averaging, for which the mean flow value is defined as

$$\overline{\phi} = \lim_{\Delta t \to \infty} \frac{1}{\Delta t} \int_t^{t+\Delta t} \phi dt$$

In practice $\Delta t \to \infty$ means that the integration time Δt needs to be long enough relative to the maximum period of the assumed fluctuations; hence, the assumption is made that the unsteady phenomena of fluid dynamics to be modelled have frequency ranges outside the frequency range of turbulence. By time averaging the Navier-Stokes equations in such a manner, the Reynolds-Averaged Navier-Stokes (RANS) equations are obtained. For compressible flows the averaging process leads to fluctuations between ρ and other variables. In order to avoid their explicit occurence, a density-weighted Favre average can be introduced

$$\phi = \tilde{\phi} + \phi''$$

where $\tilde{\phi}$ and ϕ'' are defined such that

$$\tilde{\phi} = \frac{\overline{\rho\phi}}{\overline{\rho}} \qquad \overline{\rho\phi''} = 0$$

This will remove all additional products of density fluctuations with other fluctuating quantities. Performing this averaging on the Navier-Stokes equations leads to

$$\begin{aligned}
\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial(\overline{\rho}\tilde{u}_i)}{\partial x_i} &= 0 \\
\frac{\partial(\overline{\rho}\tilde{u}_i)}{\partial t} + \frac{\partial(\overline{\rho}\tilde{u}_i\tilde{u}_j)}{\partial x_j} + \frac{\partial \overline{p}}{\partial x_i} &= \frac{\partial}{\partial x_j}(\overline{\tau_{ij}} + \tau_{ij}^R) \\
\frac{\partial(\overline{\rho}E)}{\partial t} + \frac{\partial}{\partial x_j}[\tilde{u}_j(\overline{\rho}E + \overline{p})] &= \frac{\partial}{\partial x_j}(\tilde{u}_i(\overline{\tau_{ij}} + \tau_{ij}^R) - (q_j + q_j^R))
\end{aligned} \tag{2.2}$$

where τ_{ij}^R is the Reynolds stress tensor and q_j^R is the turbulent heat flux. The additional terms, τ_{ij}^R and q_j^R, consist of products of the fluctuating components; the presence of these components is the cause of the closure problem of turbulence. Following the eddy viscosity hypothesis, which is a first order closure model, and Boussinesq's Approximation, the Reynolds stress tensor can be formulated as

$$\tau_{ij}^R = \mu_t \left[\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3}\delta_{ij}\frac{\partial \tilde{u}_k}{\partial x_k} \right] - \frac{2}{3}\overline{\rho}\delta_{ij}k$$

where μ_t is the turbulent eddy viscosity and k is the specific turbulent kinetic energy. Similarly, the turbulent heat flux vector can be modelled by

$$q_j^R = -\kappa_t \frac{\partial T}{\partial x_j}$$

where κ_t is the turbulent thermal conductivity, given by

$$\kappa_t = \frac{\mu_t C_p}{Pr_t}$$

where Pr_t is the turbulent Prandtl number, which is a constant set to 0.9.

Introducing these expressions into the RANS equations, Eq. (2.2), leads to a system that is formally identical to the Navier-Stokes equations, with μ, κ and p substituted with an effective viscosity μ_e, effective thermal conductivity κ_e, and modified pressure p^*, given by

$$\begin{aligned} \mu_e &= \mu + \mu_t \\ \kappa_e &= \kappa + \kappa_t \\ p^* &= p + \frac{2}{3}\bar{\rho}k \end{aligned} \tag{2.3}$$

Neglecting the turbulent terms μ_t and k, leads to the Navier-Stokes equations for laminar flows. It only remains to obtain relations for the turbulent terms as a function of the mean flow quantities; this is done through the use of a turbulence model. Such models are generally based on theory and empirical information; and the first order models can be classified according to the number of additional equations used. Many turbulence models are available [93]; though in this work the one equation Spalart-Allmaras model [94] is used, in which a single transport equation is solved for a quantity $\tilde{\nu}$, which is closely related to the eddy viscosity; the terms involving k are neglected.

2.1.2 Euler equations

The Euler equations, which constitute the highest inviscid modelling level, are obtained by neglecting the shear stresses and heat conduction terms from the RANS equations

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} &= 0 \\ \frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} + \frac{\partial p}{\partial x_i} &= 0 \\ \frac{\partial (\rho E)}{\partial t} + \frac{\partial}{\partial x_j}[u_j(\rho E + p)] &= 0 \end{aligned} \tag{2.4}$$

In constrast to the Navier-Stokes equations, these equations are a set of first order partial differential equations. This means that the number of allowable boundary con-

ditions is modified. For the Euler equations there is only no normal component of velocity on the solid walls.

2.1.3 Domain discretisation

The form of the Navier-Stokes equations solved in this work include the following:

- laminar Navier-Stokes Equations in non-dimensional form, Eq. (2.1);
- RANS Equations in non-dimensional form, Eq. (2.2);
- and Euler Equations in non-dimensional form Eq. (2.4).

For ease of notation, these equations can be written in a simpler vector form

$$\frac{\partial \mathbf{w}}{\partial t} + \nabla.(\mathbf{f}^i - \mathbf{f}^v) = 0 \tag{2.5}$$

where $\mathbf{w}$ is the vector of conserved variables, $\mathbf{f}^i$ is the inviscid flux vector and $\mathbf{f}^v$ is the viscous flux vector. These vectors and a summary of the Navier-Stokes equations are given in Appendix A.

In order to carry out a computational simulation of the non-linear partial differential equation in Eq. (2.5), a numerical discretisation is needed to transform it to a set of algebraic equations that can be solved. In finite volume methods the integral form of the Navier-Stokes equations is solved; and discretisation is achieved through the use of a mesh, which divides the domain Ω into a set of non-overlapping control volumes or cells Ω_i, $i = 1, \ldots, N$, where N is the number of cells. The discrete form of the integral in each cell is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}(\Omega_i \mathbf{w}) + \sum_j (\mathbf{f}^i - \mathbf{f}^v).\mathbf{n}_{ij} ds = 0 \tag{2.6}$$

where the sum of the flux terms refers to the internal sides (denoted s) of the control cell Ω_i, and $\mathbf{n}_{ij}$ is the associated outer normal vector. This discretisation means that the flux leaving one cell must enter the adjacent cell, and as such conservation is ensured.

It is for the user to decide exactly how the computational domain is to be split into the various cells; some of the problems associated with mesh generation and their application in moving-body simulations have been outlined in Chapter 1. The meshless method is used in this thesis, which instead uses points to discretise the domain. These points must be located to efficiently resolve the flow features of the problem to be solved; so to solve the viscous Navier-Stokes equations the point distributions are required to be very stretched, or anisotropic, to capture boundary-layers and wake regions. The method of determining functions to approximate the flow variables as solution to the various forms of the Navier-Stokes equations are presented in the subsequent sections of this chapter.

2.2 Meshless method

2.2.1 Overview of the meshless method

An open bounded domain $\Omega \in \mathbb{R}^D$ is discretised by a set of N points, distributed throughout Ω. To each point $\mathbf{x}_i$, $i = 1, \ldots, N$ we assign a subdomain, often called a stencil or cloud Ω_i, which contains x_i, called the "star point" of the subdomain, and a set of $(n_i - 1)$ neighbouring points, which we label j. One may use all available data points to construct each subdomain, but for the purpose of efficiency it is reasonable to select some small subset of data points in the neighbourhood of i, so that $\sum_{i=1}^{N} \Omega_i$ represents a covering of Ω. The properties of the local stencils, both required and ideal, are outlined in Chapter 5, which concerns the selection of stencil points from overlapping point distributions.

Assuming that we are given data $\{(\mathbf{x}_i, \phi(\mathbf{x}_i))\}_{i=1}^{N} \subset \mathbb{R}^D$ at each of these point data sites, where ϕ is some (smooth) function, we seek a local discretised or approximated solution $\hat{\phi}$ of the function ϕ within Ω_i. The choice of space of $\hat{\phi}$ is a vector space with simple basis. There are m basis functions, which we use to approximate the function ϕ using a linear combination of these basis functions represented by

$$\hat{\phi}(\mathbf{x}_j) = \sum_{k=1}^{m} p(\mathbf{x}_j)_k \alpha_k = \mathbf{p}^T \boldsymbol{\alpha} \qquad \forall \mathbf{x}_j \in \Omega_i \tag{2.7}$$

where $\boldsymbol{\alpha}$ is the local vector of coefficients that must be determined, and $\mathbf{p}$ is the vector of base monomials at each point j within the subdomain. The function $\hat{\phi}$ thus takes the form of a simple polynomial, so if a linear basis, such that $m = 3$, is chosen for two-dimensional calculations, then

$$\begin{aligned} \mathbf{p}(\mathbf{x}_j) &= \{1, x_j, y_j\}^T \\ \boldsymbol{\alpha} &= \{\alpha_1, \alpha_2, \alpha_3\}^T \end{aligned}$$

A bilinear polynomial, such that $m = 4$, is obtained by inclusion of the cross term

$$\begin{aligned} \mathbf{p}(\mathbf{x}_j) &= \{1, x_j, y_j, x_j y_j\}^T \\ \boldsymbol{\alpha} &= \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}^T \end{aligned}$$

Finally, if a quadratic basis is chosen, then $m = 6$, and so

$$\begin{aligned} \mathbf{p}(\mathbf{x}_j) &= \{1, x_j, y_j, x_j y_j, x_j^2, y_j^2\}^T \\ \boldsymbol{\alpha} &= \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\}^T \end{aligned}$$

The vectors in three-dimensions follow in the same way, with the inclusion of the z component. When the required basis is chosen, we must solve a linear system of n_i

equations for $\boldsymbol{\alpha}$ in each subdomain, given by

$$\hat{\boldsymbol{\phi}}_{(n_i \times 1)} = X_{(n_i \times m)} \boldsymbol{\alpha}_{(m \times 1)} \tag{2.8}$$

where X is formed from the vectors of base monomials

$$X = \{\mathbf{p}(\mathbf{x}_1), \mathbf{p}(\mathbf{x}_2), ..., \mathbf{p}(\mathbf{x}_{n_i})\}^T$$

Note that Eq. (2.8) consists of a matrix of known elements and two vectors of unknowns. The coefficients needed to form $\hat{\phi}$ within Ω_i are obtained by imposing the condition

$$\hat{\phi}(\mathbf{x}_j) = \phi(\mathbf{x}_j) \qquad \forall j \in \Omega_i \tag{2.9}$$

so at each of the nodes within the cloud, we require the approximation function to give the exact values of ϕ. To find the coefficient vector we require $n_i \geq m$, so we have at least as many equations as unknowns: this results in a well posed problem; otherwise, an infinite number of functions would be able to approximate the data equally well. An overdetermined system, such that $n_i > m$, is preferred; so we effectively have to calculate the surface of best fit for the local values of $\phi(\mathbf{x}_j) \in \Omega_i$. The use of an overdetermined system means, however, that the local values of the approximating function will not fit the values of $\phi(\mathbf{x}_j)$ at the data points: not even for the star point. More formally, it does not possess the delta property, that is

$$\hat{\boldsymbol{\phi}}(x_j) \neq \phi(x_j) \qquad j \in \Omega_i$$

The function that best approximates the exact vales at the data sites will be that for which the residual, which is the difference between $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$ measured in the $L2$ norm, is a minimum. This is the least squares problem, which can be solved either by using singular value decomposition (SVD), or by solving the normal equations, which are written as

$$X^T X \boldsymbol{\alpha} = X^T \boldsymbol{\phi} \tag{2.10}$$

The matrix $X^T X$ is called the least squares matrix; some of its properties, and the derivation of the normal equations, are given in Appendix B. The notation of the normal equations will be used in this chapter to outline the meshless method.

2.2.2 Weighting

For more accurate results, each of the equations within the system must be properly weighted so that the least squares approximation is enhanced in the vicinity of the star point. This means that points further from the star point have less influence on the approximation than those that are closer. For continuity, we should define the weight

function so that it will go to zero smoothly outside the subdomain. The weighting function must then be continuous and differentiable in Ω_i

$$\begin{aligned} w_i(\mathbf{x}_j) &> 0 \qquad \forall \mathbf{x}_j \in \Omega_i \\ w_i(\mathbf{x}) &= 0 \qquad \forall \mathbf{x} \in \Omega \backslash \Omega_i \\ w_i(\mathbf{x}_i) &= 1 \end{aligned}$$

In this work, a normalised Gaussian function [55], which was used for finite point methods in Ref. [52] and the works following, is used, which is defined by

$$w_i(\mathbf{x}_j) = \frac{e^{-(d_i/\kappa)^2} - e^{-(\beta/\kappa)^2}}{1 - e^{-(\beta/\kappa)^2}} \tag{2.11}$$

where $d_i = |\mathbf{x}_j - \mathbf{x}_i|_2$; and $\beta = \gamma d_{max}$ and $\kappa = \beta/\omega$ are constants. Introducing the weights into the system means that we effectively alter the least squares problem, and so Eq. (2.10) becomes

$$X^T W X \boldsymbol{\alpha} = X^T W \boldsymbol{\phi} \tag{2.12}$$

where $W = diag(w_i(\mathbf{x}_j))$. The significance of these constants was explored in Ref. [55]. The parameter γ increases or decreases the size of the support of the weighting function. It has only a small effect on the approximation when $n_i \approx m$; and this effect becomes negligible when this number is increased; thus, the parameter is set to a constant value $\gamma = 1.01$ throughout the domain. The parameter ω, however, can modify the local character of the least squares matrix obtained; it is plotted for three values in Fig. 2.1.

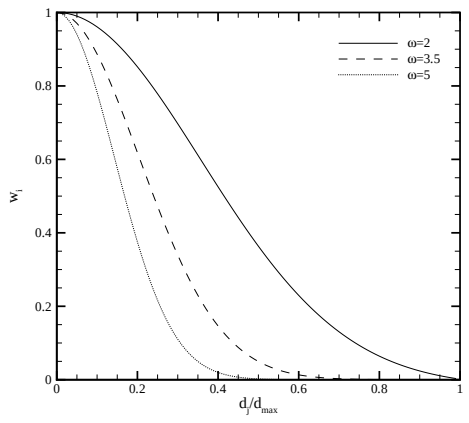


Figure 2.1: Effect of parameter ω on the weighting function Eq. (2.11) with γ=1.01.

As ω is increased, the function tends towards the Dirac delta function, and so the contribution from the neighbouring points in the stencil becomes less; consequently, the numerical error decreases, while the least squares matrix becomes more ill-conditioned. There is a threshold beyond which the ill-conditioning becomes relevant, which occurs at $\omega \approx 5$, though the exact value is different for each stencil; if ω is greater than this value then it becomes impossible to solve the normal equations. As a result, the range for ω recommended in Ref. [55] is $3.0 \leq \omega \leq 4.5$; and so, in this work, $\omega = 3.1$ which is within this range. Weighting is also beneficial to the convergence of the implicit, linear system, which is discussed in Chapter 3, as it improves the conditioning of the matrix by reducing the influence of the off-diagonal terms.

2.2.3 Shape function

The matrix $X^T W X$ is symmetric and positive-definite (Appendix B); this property can be exploited by the use of a Cholesky decomposition to solve Eq. (2.12). This is an efficient, stable matrix algorithm, which requires no pivoting, and is about a factor of two faster than alternative methods for solving linear systems due to the matrix symmetry [95]. The matrix is decomposed into the product of a lower triangular matrix and its transpose; a back substitution is then performed to obtain the solution.

If the solution $\boldsymbol{\alpha}$ can be found accurately enough (that is, the matrix is well-conditioned) then the approximate function, from Eqs. (2.8) and (2.12), can be written

$$\begin{aligned} \hat{\boldsymbol{\phi}} &= X\boldsymbol{\alpha} \\ &= X(X^T W X)^{-1} X^T W \boldsymbol{\phi} \\ &= N\boldsymbol{\phi} \end{aligned}$$

where N is an $(n_i \times n_i)$ matrix, which determines the approximate function for each point in the cloud.

As one point within the subdomain can belong to several other overlapping subdomains, each of these subdomains can yield different shape functions for the same point; thus, we only find the approximate function $\hat{\phi}$ at specific regions within each subdomain. Using point collocation we in fact limit the validity of the interpolation within the subdomain to *only* the star point [55], as opposed to every point in the stencil. Then, using Eq. (2.12) in Eq. (2.7), so the approximation is only at the star, the approximation within the stencil can be written

$$\begin{aligned} \hat{\phi}_i(\mathbf{x}_i) &= \mathbf{p}_i^T(\mathbf{x}_i)\boldsymbol{\alpha} \\ &= \mathbf{p}_i^T(\mathbf{x}_i)(X^T W X)^{-1} X^T W \boldsymbol{\phi} \\ &= \mathbf{N}_i(\mathbf{x}_i)\boldsymbol{\phi} \end{aligned} \tag{2.13}$$

where $\mathbf{N}_i$ is a $(1 \times n_i)$ vector called the shape function of point $\mathbf{x}_i \in \Omega_i$. Note the similarity of Eq. (2.13) to Eq. (1.1) for determining approximate functions using meshless methods; the non-reciprocal nature mentioned in Chapter 1 holds for Eq. (2.13).

The shape function is of fundamental importance in the meshless method; it is unique for the star point within each subdomain Ω_i, and can be written explicitly as

$$\mathbf{N}_i = \mathbf{p}^T(X^T W X)^{-1} X^T W \tag{2.14}$$

The shape function is dependent only on the relative position of the points, so if the geometry of the points remains the same, as is the case for stationary problems, then it is sufficient to calculate Eq. (2.14) once and store the values in data structures to reduce the computational costs.

2.2.4 Mapping

It is possible for the conditioning of the least squares matrix to be very poor, which can lead to inaccuracies in the shape function computation. This is especially true if the stencils are highly stretched in nature. The aspect ratio for some stencils can be of size 10^{-3}–10^{-6}, which are necessary to capture the flow features that occur in fluid dynamics, as mentioned in Section 2.1.3. Furthermore, recall that the weights reduce the influence of points within the stencil that are further away from the star point. For a highly directional stencil, this can make the influence of several of the points negligible, which can result in an approximation equivalent to $n_i < m$, with the remaining points lying in a straight line: resulting in a singular least squares matrix. We can resolve this by mapping the stencil distribution to a new computational domain, finding the shape functions, and then mapping back to the physical domain.

The new computational domain for the stencil is obtained by translating the cloud so that the star point is located at the origin. We then measure the distance between the star point and each of its neighbours, located at $\mathbf{x}'$ along each of the coordinate axes, and store the largest absolute values; these are denoted R_x, R_y and R_z. The coordinates within the stencil are then normalised within a new coordinate system $\boldsymbol{\zeta}$, such that

$$\zeta_1 = \frac{x'}{R_x} \qquad \zeta_2 = \frac{y'}{R_y} \qquad \zeta_3 = \frac{z'}{R_z}$$

The shape functions are then computed in this system to give

$$\mathbf{N}_i(\boldsymbol{\zeta}_j) = \mathbf{p}^T(\boldsymbol{\zeta}_j)((X^T W X)^{-1} X^T W)(\boldsymbol{\zeta}_j)$$

where the weighting function within these matrices is written

$$w_i(\boldsymbol{\zeta}_j) = \frac{e^{-\left(\frac{d_i}{\kappa}\right)^2} - e^{-\left(\frac{\beta}{\kappa}\right)^2}}{1 - e^{-\left(\frac{\beta}{\kappa}\right)^2}}$$

where $d_i = \sqrt{\zeta_{1j}^2 + \zeta_{2j}^2 + \zeta_{3j}^2}$. This system will be much easier to solve, and will improve the conditioning of the least squares matrix. When the solution is obtained, the shape function derivatives are then mapped back to the physical domain to give Eq. (2.14) for use by the solver.

2.2.5 Derivatives of the approximated function

To solve a partial differential equation in the form of a conservation law, we need to use the meshless method to compute the derivatives of the functions. This is done by applying derivatives to the polynomial obtained using the least squares method, and assigning these polynomial derivatives as approximations to the derivative of the function. Using the Einstein summation notation and applying the product rule directly

to Eq. (2.13), we can write that for each star point $\mathbf{x}_i \in \Omega$, the following approximation for the derivative of $\hat{\phi}_i$ with respect to x is

$$\begin{aligned}
\frac{\partial \hat{\phi}_i}{\partial x} &= \frac{\partial}{\partial x}(N_{(i)j}\phi_j) \\
&= \frac{\partial}{\partial x}\left\{N_{(i)j}\right\}\phi_j \\
&= \frac{\partial}{\partial x}\left\{p_{(i)k}((X^TWX)^{-1}X^TW)_{kj}\right\}\phi_j \qquad\qquad k = 1, ..., m \\
&= \frac{\partial}{\partial x}\left\{p_{(i)k}\right\}\left\{((X^TWX)^{-1}X^TW)_{kj}\right\}\phi_j + \left\{p_{(i)k}\right\}\frac{\partial}{\partial x}\left\{((X^TWX)^{-1}X^TW)_{kj}\right\}\phi_j \\
&\approx \frac{\partial}{\partial x}\left\{p_{(i)k}\right\}\left\{((X^TWX)^{-1}X^TW)_{kj}\right\}\phi_j \\
&= ((X^TWX)^{-1}X^TW)_{2j}\phi_j \\
&= b_{(i)j}\phi_j
\end{aligned} \tag{2.15}$$

where b_i represents the $(1 \times n_i)$ shape function derivatives with respect to x at i. Note that we have simplified the approximation and computational cost considerably by assuming that the matrix X is constant in Ω_i. This distinguishes the method from the moving least squares approach, and is often called the fixed least squares approach [52]. The functions obtained are now discontinuous, and hence the consistency conditions (see next section) can only be satisfied for a point within the subdomain of which it is the star point [55]. This is in keeping with the point collocation method as described in Section 2.2.3. The derivatives with respect to y and z follow in the same way; hence, the approximate function derivatives using the meshless method can be obtained from

$$\frac{\partial \hat{\phi}_i}{\partial x} = \sum_{j \in \Omega_i} b_{(i)j}\phi_j \qquad \frac{\partial \hat{\phi}_i}{\partial y} = \sum_{j \in \Omega_i} c_{(i)j}\phi_j \qquad \frac{\partial \hat{\phi}_i}{\partial z} = \sum_{j \in \Omega_i} d_{(i)j}\phi_j \tag{2.16}$$

where c_i and d_i are the shape function derivatives with respect to y and z at i respectively. The similarity of these equations to Eq. (2.13) means that the terms shape function and shape function derivatives are often used interchangably in meshless works.

2.2.6 Partition of unity

We can show that the shape function $\mathbf{N}_i$ is a partition of unity function, which allows us to patch together the approximate functions $\hat{\phi}_i$ found within each subdomain locally, to give a global solution [64]. If this is the case, then within each subdomain Ω_i we choose an approximation space V, then a function ϕ can be approximated within Ω_i by a local approximant $p_i(x_j) \in V$; the local approximants are put together by forming

$$p_i(x) = \sum_{j=0}^{n_i} p_i(x_j)N_{(i)j}(x) \tag{2.17}$$

The partition of unity means that any function of order k in the basis, complete in the polynomials of order k, can be reproduced exactly. In the literature this is referred to as the "consistency" of the approximation [45]. For example, if the basis is linear then the shape functions will satisfy linear consistency; without this condition we would not be able to form a global approximation within the entire domain Ω. The proof is from Ref. [64], and follows straight from the explicit form of the shape function in Eq. (2.14), which we state as

$$\begin{aligned}
\sum_{j=0}^{n_i} p_i(x_j) N_{(i)j}(x) &= \sum_{j=0}^{n_i} p_i(x_j) \sum_{k,l=1}^{m} p_l(x) A^{-1}_{lk}(x) p_k(x_j) \\
&= \sum_{k,l=1}^{m} \sum_{j=0}^{n_i} p_i(x_j) p_k(x_j) A^{-1}_{lk}(x) p_l(x) \\
&= \sum_{k,l=1}^{m} A_{ik}(x) A^{-1}_{lk}(x) p_l(x) \\
&= p_i(x)
\end{aligned}$$

as required by Eq. (2.17). Polynomials are used in the definition of the approximating functions, and from Section 2.2.1 we can see that $1 \in p_i(x_j)$. This zero order polynomial ($m = 1$) implies that

$$\sum_{j=0}^{n_i} 1.N_{(i)j}(x) = \sum_{j=0}^{n_i} N_{(i)j}(x) = 1$$

which is the partition of unity property often quoted [1]. It can be shown that any function which appears in the basis can be reproduced exactly.

To compute derivatives of $\hat{\phi}_i$, the shape functions must satisfy the partition of nullity to ensure k order consistency, for which the following condition must be satisfied

$$\sum_{j=0}^{n_i} \frac{\partial^k N_{(i)j}(x)}{\partial x^k} p_i(x_j) = \frac{\partial^k p_i(x)}{\partial x^k}$$

Using the notation from the previous section, with a first order derivative, so that $k = 1$, then

$$\sum_{j=0}^{n_i} b_{(i)j} p_i(x_j) = \frac{\partial p_i(x)}{\partial x}$$

Once again if we use the case $1 \in p_i(x_j)$, as above, then

$$\sum_{j=0}^{n_i} b_{(i)j}.1 = \sum_{j=0}^{n_i} b_{(i)j} = 0 \tag{2.18}$$

[1] That the set of functions $p_i(x) \in C^\infty$ is also a property, however, this is not essential in constructing meshless approximations; in fact we have already shown that the functions are not continuous.

This property must be correct for a meshless approximation to be made; for example, if we consider a constant function within a stencil, then all of the points have the same value of ϕ, then from Eq. (2.16) and Eq. (2.18)

$$\frac{\partial \hat{\phi}_i}{\partial x} = \phi_j \left(\sum_{j=0}^{n_i} b_{(i)j} \right) = 0$$

which means that the spatial derivatives will be zero for this stencil, as required.

Equation (2.18) means that the following relation for the shape function derivative at the star point holds

$$b_{(i)j} = -\sum_{j \neq i} b_{(i)j}$$

This means that we can reduce the dimension of the least squares matrix by one, by neglecting the zero order polynomial term. This method reduces the computational costs of the scheme, which is particularly important if higher order polynomials are to be used, or there are repeated shape function calculations as is the case in moving-body simulations.

2.3 Discretisation of the Navier-Stokes equations using the meshless method

2.3.1 Evaluating the inviscid flux

The Euler equations are a simplification of the full Navier-Stokes equations, formed by removing the viscous and heat-conduction terms. They are a set of hyperbolic conservation laws, which can be written in conservative form and Cartesian coordinates as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{w})}{\partial x} + \frac{\partial \mathbf{g}(\mathbf{w})}{\partial y} + \frac{\partial \mathbf{h}(\mathbf{w})}{\partial z} = 0 \tag{2.19}$$

where $\mathbf{w}$ denotes the vector of conserved variables, which is to be determined, and $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{h}$ are the inviscid flux vectors. To solve this system using the meshless method, we write Eq. (2.19) for each local subdomain i in discrete form, using Eq. (2.16) to approximate the spatial flux derivatives, such that

$$\frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} = -\sum_{j \in \Omega_i} \left(b_{(i)j}\mathbf{f}_j + c_{(i)j}\mathbf{g}_j + d_{(i)j}\mathbf{h}_j \right)$$

This form of the discretisation is centred, and as such is unstable for use with hyperbolic partial differential equations, for which information propagates along characteristics. Stability is obtained if we use an upwind scheme for the flux functions defined halfway between the star point i and the neighbouring point j (analogous to the flux at the

face of a control volume in a finite volume scheme), at $j-\frac{1}{2}$; this leads to an equivalent discrete expression

$$\frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} = -\sum_{j\in\Omega_i}\left(b_{(i)j-\frac{1}{2}}\mathbf{f}_{j-\frac{1}{2}} + c_{(i)j-\frac{1}{2}}\mathbf{g}_{j-\frac{1}{2}} + d_{(i)j-\frac{1}{2}}\mathbf{h}_{j-\frac{1}{2}}\right) \tag{2.20}$$

From Eq. (2.20) we can see that the non-reciprocal nature of the shape function derivatives means that conservation is not ensured in the same way as for finite volume methods.

Instead of computing the flux at i and j directly, the computation at the location halfway across the edge can be seen as an interface between the flux values at i and those at j, which are approximated as constant. The flux is calculated by solving a Riemann problem [96]. The Riemann problem can be solved approximately using Osher's method [97], which has the convenient property of being continuously differentiable, or Roe's method [98] with an appropriate entropy correction, for which the mid-point flux is written as

$$\mathbf{f}_{j-\frac{1}{2}} = \frac{1}{2}(\mathbf{f}(\mathbf{p}_L) + \mathbf{f}(\mathbf{p}_R)) - \frac{1}{2}\left|\tilde{A}(\mathbf{p}_L, \mathbf{p}_R)\right|(\mathbf{p}_R - \mathbf{p}_L) \tag{2.21}$$

where $\tilde{A}$ is the Roe averaged Jacobian matrix, and $\mathbf{p}_L$ and $\mathbf{p}_R$ are the vectors of primitive variables at the left and right hand sides of the interface at the halfway position. For a first order scheme the left and right states are simply the values of the flow variables located at the star and neighbouring points

$$\begin{aligned} \phi_L &= \phi_i \\ \phi_R &= \phi_j \end{aligned} \tag{2.22}$$

Denote the number of points that the local flux depends on as $\mathcal{M}(i)$; then for a first order scheme $\mathcal{M}(i) = n_i$. Practical engineering calculations, however, generally require at least second order spatial accuracy, which is achieved in the same way as in many unstructured finite volume codes [99]. The left and right states of the Riemann problem are obtained by extrapolating the values at i and j, based on a reconstructed gradient given by

$$\begin{aligned} \phi_L &= \phi_i + \psi_i \mathbf{s}_{ij}.\nabla\phi_i \\ \phi_R &= \phi_j - \psi_j \mathbf{s}_{ij}.\nabla\phi_j \end{aligned} \tag{2.23}$$

where $\mathbf{s}_{ij} = \frac{1}{2}(\mathbf{x}_j - \mathbf{x}_i)$. These gradients are calculated at each point using the shape functions of Eq. (2.16), so are not evaluated halfway across. Using this scheme, the flux now depends on the stencils of the neighbouring points as well (as they form $\nabla\phi_j$), so $\mathcal{M}(i) > n_i$. To counter oscillations near discontinuities, flux limiters, ψ_i and ψ_j, are used, which bound the reconstructed function of each flow variable between the

extreme values within the stencil by reducing the scheme to first order whenever these oscillations may occur; there are several limiters available for use in this work.

The Barth-Jespersen limiter [100] was the first limiter used for unstructured grids; the formulation is as follows

1. Find the largest negative ($\delta\phi_i^{min} = \min_{j\in\Omega_i}(\phi_j - \phi_i)$) and positive ($\delta\phi_i^{max} = \max_{j\in\Omega_i}(\phi_j - \phi_i)$) difference between the solution at the points $j \in \Omega_i$ and the star point i of the current stencil.

2. Compute the unconstrained reconstructed value at each point ($\phi_{ij} = \phi_i + \mathbf{s}_{ij}.\nabla\phi_i$).

3. Compute a maximum allowable value of ψ_{ij} for each point

$$\psi_{ij} = \begin{cases} \min(1, \frac{\delta\phi_i^{max}}{\phi_{ij}-\phi_i}), & \text{if } \phi_{ij} - \phi_i > 0 \\ \min(1, \frac{\delta\phi_i^{min}}{\phi_{ij}-\phi_i}), & \text{if } \phi_{ij} - \phi_i < 0 \\ 1, & \text{if } \phi_{ij} - \phi_i = 0 \end{cases}$$

4. Select $\psi_i = \min(\psi_{ij})$

This limiter is very strong and prevents new local extrema being formed during reconstruction; however, it is non-differentiable, which can have an adverse effect on the convergence rate of the solver. For this reason the Venkatakrishnan limiter [101] is also available, which is a smooth alternative to the Barth-Jespersen limiter and is stated as follows

$$\psi_i = \frac{(\Delta_j^+)^2 + 2\Delta_j^+\Delta_j^- + \beta}{(\Delta_j^+)^2 + 2(\Delta_j^-)^2 + \Delta_j^+\Delta_j^- + \beta}$$

where $\Delta_j^- = \mathbf{s}_{ij}.\nabla\phi_i$ and, using the same notation as step 1 for the Barth-Jespersen scheme,

$$\Delta_j^+ = \begin{cases} \delta\phi_i^{max} - \phi_i, & \text{if } \Delta_j^- > 0 \\ \delta\phi_i^{min} - \phi_i, & \text{if } \Delta_j^- < 0 \end{cases}$$

The parameter $\beta = (\kappa h)^3$ is a constant for each stencil, where h can be taken to be the minimum length in the stencil connectivity, and κ is a tunable parameter that determines the amount of limiting, with a larger value causing less limiting. Tests with PML, and in agreement with Ref. [58], show that $\kappa = 3$ gives the best results. Finally, the Van Albada limiter, commonly used with MUSCL type schemes in meshless solvers [79], is implemented, for which ψ takes the form

$$\begin{aligned} \psi_i &= \max\left\{0, \frac{2\Delta_i^-(\phi_j - \phi_i) + \epsilon}{(\Delta_i^-)^2 + (\phi_j - \phi_i)^2 + \epsilon}\right\} \\ \psi_j &= \max\left\{0, \frac{2\Delta_j^+(\phi_j - \phi_i) + \epsilon}{(\Delta_j^+)^2 + (\phi_j - \phi_i)^2 + \epsilon}\right\} \end{aligned}$$

where ϵ is a small tolerance included to prevent division by zero, which can occur in smooth regions of flow; and Δ_i^- and Δ_j^+ are forward and backward difference operators given by

$$\begin{aligned} \Delta_i^- &= 2\mathbf{s}_{ij}.\nabla\phi_i - (\phi_j - \phi_i) \\ \Delta_j^+ &= 2\mathbf{s}_{ij}.\nabla\phi_j - (\phi_j - \phi_i) \end{aligned}$$

This limiter is free of parameters, and there is no need to evalutate minimum and maximum values as is required in the other two limiters stated; however, it is slightly more dissipative. The results presented in this thesis use the Barth-Jespersen limiter for calculations in two-dimensions, while the Van Albada limiter is used in three-dimensions; the reasons for this will be seen in Chapter 4.

2.3.2 Evaluating the viscous flux

The Navier-Stokes equations require the addition of the viscous fluxes to the Euler equations; thus, Eq. (2.19) is modified to

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{f} - \mathbf{f}^v) + \frac{\partial}{\partial y}(\mathbf{g} - \mathbf{g}^v) + \frac{\partial}{\partial z}(\mathbf{h} - \mathbf{h}^v) = 0 \tag{2.24}$$

where $\mathbf{f}^v$, $\mathbf{g}^v$ and $\mathbf{h}^v$ are the viscous fluxes in the x, y and z directions. In some respects the discretisation of the viscous terms is not as problematic as for the inviscid terms, as they are parabolic in nature, and so do not require any upwinding. Instead, a variation on the central difference scheme is sufficient for these terms. To calculate the fluxes at the interface between the points, the flow variables are simply averaged

$$\phi_{ij} = \frac{1}{2}(\phi_i + \phi_j)$$

Unlike the inviscid fluxes, which contain only the flow quantities, the viscous fluxes also contain the derivatives of the flow quantities, as seen in the stress tensors. There are two ways to include these terms in the discretisation: one is to separate the inviscid and viscous flux terms, and use second order derivatives obtained directly from the least squares approximation in the stress tensor computations; the other is to use the first order gradients directly in the tensors. The first method means that it is necessary for the approximation polynomials to be of at *least* quadratic order in Eq. (2.7). As such, the stencils must be larger for every point in the domain: there is no room for flexibility. It also means that second order shape functions must be computed and stored for use in calculating the additional second order gradients. Instead, the second method is employed in this work, which is easier to implement and is more cost effective to evaluate. The gradients used in the flux are obtained by a simple averaging of the

gradients at i and j, to give the midpoint gradient

$$\nabla\phi_{ij} = \frac{1}{2}(\nabla\phi_i + \nabla\phi_j)$$

The advantage of this method is that the same gradient values as in Eq. (2.23) are used; the disadvantage is that odd-even node decoupling is not suppressed. To overcome this, the derivative quantities that appear in the viscous flux can be constructed using a modified gradient [99], defined as

$$\nabla\phi_{ij} = \frac{1}{2}(\nabla\phi_i + \nabla\phi_j) - \left[\frac{1}{2}(\nabla\phi_i + \nabla\phi_j).\frac{\mathbf{t}_{ij}}{|\mathbf{t}_{ij}|} - \frac{\phi_j - \phi_i}{|\mathbf{t}_{ij}|}\right].\frac{\mathbf{t}_{ij}}{|\mathbf{t}_{ij}|} \tag{2.25}$$

where $\mathbf{t}_{ij} = (\mathbf{x}_j - \mathbf{x}_i)$ is the vector joining points i and j. The flux derivatives of these midpoint terms are then found by using the same shape function derivatives as for the inviscid flux, evaluated halfway between i and j; hence, the meshless discretisation of Eq. (2.24) is given by

$$\frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} = -\sum_{j\in\Omega_i}\left(b_{(i)j-\frac{1}{2}}(\mathbf{f}_{j-\frac{1}{2}} - \mathbf{f}^v_{ij}) + c_{(i)j-\frac{1}{2}}(\mathbf{g}_{j-\frac{1}{2}} - \mathbf{g}^v_{ij}) + d_{(i)j-\frac{1}{2}}(\mathbf{h}_{j-\frac{1}{2}} - \mathbf{h}^v_{ij})\right) \tag{2.26}$$

As the local flow variables and gradients at each point in the stencil are used to determine the viscous flux, then $\mathcal{M}(i)$ is the same as for the second order inviscid flux.

2.3.3 Evaluating the turbulence model terms

As discussed in Section 2.1.1, high Reynolds number flows are resolved by Reynolds-Averaging the Navier-Stokes equations, with the addition of a turbulence model to close the resultant system. The RANS equations are identical to the Navier-Stokes equations, with μ, κ and p replaced by the corresponding effective terms given in Eq. (2.3). Using these terms in the numerical scheme is the simplest way to introduce turbulence through the RANS equations. The unknowns in Eq. (2.3) are evaluated using the turbulence model, which in this work is the one-equation Spalart-Allmaras (SA) model [94]. In this model, a transported quantity, denoted $\tilde{\nu}$, is determined, from which the eddy viscosity μ_t is calculated using the following relation

$$\mu_t = \rho\tilde{\nu}f_{v1}$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c^3_{v1}}, \qquad \chi = \frac{\tilde{\nu}}{\nu}$$

and $c_{v1} = 7.1$ is a model constant. The quantity $\tilde{\nu}$ can be determined from the equation

$$\frac{\partial(\rho\tilde{\nu})}{\partial t} + \nabla.(\rho\tilde{\nu}\mathbf{u} - \mu^t_e\nabla\tilde{\nu}) = S \tag{2.27}$$

where S is the source term contribution and μ_e^t is the effective turbulent viscosity. The effective turbulent viscosity is not the same as the mean flow effective viscosity μ_e; for the Spalart-Allmaras model is it simply

$$\mu_e^t = \frac{1}{\sigma_m}(\mu + \mu_t)$$

where $\sigma_m = \frac{2}{3}$ is another model constant. The terms within the divergence operation of Eq. (2.27) can be viewed as flux contributions: the first being a convective flux, the second being a diffusive flux. The convective flux is discretised using the same reconstructed states between two points as for the inviscid flux of the mean flow equations in Eq. (2.23). The flux function in the x direction may be written

$$f_{j-\frac{1}{2}}^{t,conv} = \frac{1}{2}q_{ij}(\rho_i\tilde{\nu}_i + \rho_j\tilde{\nu}_j) - \frac{1}{2}|q_{ij}|(\rho_j\tilde{\nu}_j - \rho_i\tilde{\nu}_i)$$

where

$$q_{ij} = \frac{1}{2}(u_i + u_j).$$

The velocities in the y and z direction are used in q_{ij} to give the flux functions in their respective directions. Meanwhile, the diffusive flux uses averaged values between the points as in Eq. (2.25), which is the same averaging used for the evaluation of the viscous flux in the mean flow equations. The diffusive flux in the x direction may be written

$$f_{ij}^{t,diff} = \mu_e^t(\nabla_x\tilde{\nu})_{ij}.$$

where $(\nabla_x\tilde{\nu})_{ij}$ is the component of the modified, average gradient in the x direction. The diffusive flux in the y and z directions are the same, using their respective components of the gradient. The laminar and turbulent viscosities in the effective turbulent viscosity are calculated using an averaging process too

$$\begin{aligned} \mu &= \frac{1}{2}(\mu_i + \mu_j) \\ \mu_t &= \frac{1}{2}(\rho_i\tilde{\nu}_i + \rho_j\tilde{\nu}_j) \end{aligned}$$

Thus, for the flux contributions $\mathcal{M}(i)$ is the same size as for the second order inviscid, and viscous fluxes. The source term follows Edwards' modification [102], and is calculated using only the flow variables and gradients at each point; as a result $\mathcal{M}(i) = n_i$ for this term. Equation (2.27) can be solved separately from the mean equations, or they can be solved fully coupled as is done for the results presented in this work. In this case, the flux definitions above mean that the transport equation can be combined with the Navier-Stokes equations of Eq. (2.24), to give the full governing equations to be solved as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{f} - \mathbf{f}^v) + \frac{\partial}{\partial y}(\mathbf{g} - \mathbf{g}^v) + \frac{\partial}{\partial z}(\mathbf{h} - \mathbf{h}^v) = \mathbf{S} \tag{2.28}$$

where $\mathbf{S}$ is the source term vector, which is non-zero only for the turbulence transport equation. Similarly, the additional turbulence model terms can be added to Eq. (2.26) for the full meshless discretisation

$$\frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} = -\sum_{j\in\Omega_i}\left(b_{(i)j-\frac{1}{2}}(\mathbf{f}_{j-\frac{1}{2}} - \mathbf{f}^v_{ij}) + c_{(i)j-\frac{1}{2}}(\mathbf{g}_{j-\frac{1}{2}} - \mathbf{g}^v_{ij}) + d_{(i)j-\frac{1}{2}}(\mathbf{h}_{j-\frac{1}{2}} - \mathbf{h}^v_{ij})\right) + \mathbf{S}_i \tag{2.29}$$

The solution method for this discretisation, to obtain $\mathbf{w}$, is discussed in Chapter 3.

2.3.4 Boundary conditions

The numerical schemes that have been described so far are completed by the imposition of appropriate boundary conditions. For the Euler equations, the slip condition is enforced on solid wall boundaries. The points that form the boundary $\partial\Omega$ make up a set of elements that provide a covering such that the domain Ω is bounded. These elements are used to enforce the boundary conditions on the wall. To do this we need the points that make up each element e. These elements form the set $\mathcal{B}(e)$. In two-dimensions $\mathcal{B}(e)$ contains two points because each element is a line segment made up of two points; in three-dimensions $\mathcal{B}(e)$ can contain three or four points, depending on whether the element is a triangle or quadrilateral (mixed sets of elements are possible). A halo point located outside the domain is assigned to each element to impose the boundary conditions on each of these elements. The halo point is added to the stencil of each point in $\mathcal{B}(e)$, so that the flow variables set at the halo provide the required behaviour on the boundary itself. Thus, the use of halo points also increases the quality of the boundary point stencils; they ensure that there will be points in all directions in the stencil, and so the conditioning of the least squares matrix improves dramatically.

The location of each halo is such that it lies along the normal of the element from the element centre. The distance of the halo from the centre of e is determined by using the internal (non-boundary) points that are shared by the stencils of each of the points in $\mathcal{B}(e)$. We denote this set of internal points $\mathcal{C}(e)$. We take the projection of the vector formed from the centre of e to each point in $\mathcal{C}(e)$ onto the normal vector of the element. The average length of the projections determines the halo point distance. In this way the halo point distance reflects the relative size of the stencil.

The slip condition for unsteady simulations with moving surfaces requires that the flow velocity normal to the boundary is equal to the normal velocity of the wall. So for every solid wall boundary element

$$un_x + vn_y + wn_z = \dot{x}n_x + \dot{y}n_y + \dot{z}n_z \tag{2.30}$$

where n_x, n_y and n_z are the components of the inner unit normal vector at the boundary element, and the Cartesian wall velocity components are represented by $\dot{x}$, $\dot{y}$ and $\dot{z}$.

Defining the values u_e, v_e and w_e as the average of the flow velocity values at the points within the set $\mathcal{C}(e)$, the boundary condition is enforced by setting the negative normal velocity component of u_e, v_e and w_e at the halo point. Using the method in Ref. [103], this is done by a simple rotation and negation of the normal components of velocity, so that at the halo the velocity values are

$$\begin{aligned} u_h &= u_e - 2n_x u_n + 2\dot{x} \\ v_h &= v_e - 2n_y u_n + 2\dot{y} \\ w_h &= w_e - 2n_z u_n + 2\dot{z} \end{aligned}$$

where u_n is the normal component of velocity

$$u_n = u_e n_x + v_e n_y + w_e n_z$$

and the subscript h denotes the flow variables at the halo. The density and pressure halo values are set simply as the average of the values at the points in $\mathcal{C}(e)$, denoted ρ_e and p_e, so

$$\begin{aligned} \rho_h &= \rho_e \\ p_h &= p_e \end{aligned}$$

For viscous flows, the no-slip condition is imposed on the solid wall surfaces, which means that

$$u = \dot{x}, \qquad v = \dot{y}, \qquad w = \dot{z} \tag{2.31}$$

The points on the boundary wall, not the elements, are used to enforce these conditions. A strong boundary condition is employed, in which we set the flow variables u, v and w at the boundary points so that Eq. (2.31) is satisfied. To improve the stencils of the boundary points we employ halo points in the same way as in Ref. [103]. In this method, the halos are set from a reflection across the boundary wall of each internal point in the boundary stencil. As a result, there are more halo points for viscous computations than for inviscid. The flow variables at each halo are set using the flow variables at its corresponding interior point such that

$$\begin{aligned} u_h &= -u_i + 2\dot{x} \\ v_h &= -v_i + 2\dot{y} \\ w_h &= -w_i + 2\dot{z} \end{aligned}$$

where u_i, v_i and w_i are the flow velocities at the interior point. The pressure and density at the halos are set so that they have the same values as its corresponding

interior point so

$$\begin{aligned} \rho_h &= \rho_i \\ p_h &= p_i \end{aligned}$$

The boundary condition for the turbulence model is enforced so that the transported quantity $\tilde{\nu}$ is zero on the boundary walls; this condition is also imposed strongly. In addition, the halo point values are set to

$$\tilde{\nu}_h = -\tilde{\nu}_i$$

where $\tilde{\nu}_i$ is the transport quantity at the corresponding interior point of the halo.

Far field and symmetry boundary conditions are imposed on each point with one halo, which is located using a reflection across the boundary wall (in the same way as for solid, no-slip wall conditions) of one point within the stencil. The point chosen is the one for which the vector from the boundary point to the stencil point has the smallest angle to the normal vector of the boundary point. Symmetry conditions are imposed by setting the flow variables as the same as the interior point from which the reflection is made, but the component of velocity that coincides with the normal of the plane is negated. So, for example, if the symmetry plane is in the xz plane, the component of velocity in the y direction is negated

$$v_h = -v_i$$

Far field conditions are enforced by setting the halo point h to freestream values. This treatment assumes that the boundary is located far enough to allow the flow to return to freestream conditions.

Chapter 3

Time Integration

3.1 Implicit method

The meshless method is used in the spatial discretisation to find the flux derivatives, which form the residual vector $\mathbf{R}$ consisting of the right hand side of Eqs. (2.20), (2.26) or (2.29). Its definition provides us with a set of global ordinary differential equations

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -\mathbf{R} \tag{3.1}$$

with the sign on the right following convention in CFD. To solve this system we need to perform an integration with respect to time. Most meshless codes in the literature use explicit methods (with appropriate convergence accelerators) to solve Eq. (3.1); in such methods the residual vector is expressed at the current time step m, to be solved for the vector $\mathbf{w}$ at time $m+1$. In this work however, we use a fully implicit time discretisation, the implementation of which is described in this chapter.

Implicit methods are characterised by the residual vector being expressed at the unknown, new time level, creating a system of non-linear algebraic equations to be solved. To solve Eq. (3.1) to steady state, so that $\mathbf{R} = 0$, we introduce an iteration in pseudo-time τ such that

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau} = -\mathbf{R}^{m+1} \tag{3.2}$$

where the superscript m denotes the time level in pseudo-time. There are, in general, two methods that can be used to solve this system.

1. Choose some linearisation of $\mathbf{R}$, and reduce the non-linear system to a linear algebraic equation.

2. Solve the system at each time step using a non-linear sub-iteration: the dual time stepping method.

In this work the first method is used for computations to steady state, while the second method is used for time-dependent simulations. The linearisation of the global residual

vector $\mathbf{R}$, for the first case, is performed in pseudo-time τ

$$\begin{aligned}\mathbf{R}^{m+1} &= \mathbf{R}^m + \frac{\partial \mathbf{R}}{\partial \tau}\Delta\tau^m + O(\Delta\tau^2) \\ &= \mathbf{R}^m + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}\frac{\partial \mathbf{p}}{\partial \tau}\Delta\tau^m + O(\Delta\tau^2) \\ &= \mathbf{R}^m + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}\Delta\mathbf{p} + O(\Delta\tau^2) \end{aligned} \tag{3.3}$$

where $\mathbf{p}$ is the vector of primitive variables, $\Delta\mathbf{p} = \mathbf{p}^{m+1} - \mathbf{p}^m$ is the difference between these variables after each pseudo-time step, and $\frac{\partial \mathbf{R}}{\partial \mathbf{p}}$ is the Jacobian matrix of the system. It is most convenient for the global vector of primitive variables to take the form

$$\mathbf{p} = (\rho_1, u_1, v_1, w_1, p_1, \ldots, \rho_N, u_N, v_N, w_N, p_N)^T$$

where N is the total number of points in the domain. The global vector, therefore, consists of each of the local vectors of length l, where l is the number of unknown variables per point; the global residual vector must be constructed in the same way. The Jacobian matrix is formed by an analytical differentiation of the residual vector with respect to the vector of primitive flow variables, as this differentiation is simpler than if the residual vector were to be differentiated with respect to the conservative flow variables.

There are three issues that need to be addressed in determining a solution strategy using an implicit method, with the linearisation scheme outlined above: the time derivative approximation, the approximation (if any) to the Jacobian matrix of $\mathbf{R}$, and the method of solving the linear system and to what accuracy. These issues are linked in determining the accuracy, cost and stability of the solver at each step, and are discussed in the following sections.

3.2 Time derivative

The time derivative in Eq. (3.2) can be discretised with a differential operator such that

$$D_\tau \mathbf{w} = -\mathbf{R}^{m+1}$$

where D_τ is chosen to be a simple, kth order accurate, backward differential operator of the form

$$D_\tau = \frac{1}{\Delta\tau}\sum_{q=1}^{k}\frac{1}{q}[\Delta]^q \tag{3.4}$$

For time-independent calculations, for which the convergence is to steady state, the converged solution is independent of the choice of temporal discretisation, and so the simplest, first order option $k = 1$ is taken. Using this option in Eq. (3.4) in pseudo-time,

and performing the linearisation outlined above, we obtain the system to be solved for the updates $\Delta\mathbf{p}$

$$\begin{aligned}
\frac{\Delta\mathbf{w}}{\Delta\tau} &= -\left(\mathbf{R}^m + \frac{\partial\mathbf{R}}{\partial\mathbf{p}}\Delta\mathbf{p}\right) \\
\frac{\Delta\mathbf{w}}{\Delta\tau} + \frac{\partial\mathbf{R}}{\partial\mathbf{p}}\Delta\mathbf{p} &= -\mathbf{R}^m \\
\left(\frac{1}{\Delta\tau}\frac{\partial\mathbf{w}}{\partial\mathbf{p}} + \frac{\partial\mathbf{R}}{\partial\mathbf{p}}\right)\Delta\mathbf{p} &= -\mathbf{R}^m
\end{aligned} \tag{3.5}$$

where $\frac{\partial\mathbf{w}}{\partial\mathbf{p}}$ is the transformation matrix between conservative and primitive variables. The size of $\Delta\tau$ is determined by a local time step estimate [103]

$$\Delta\tau_i = \frac{\mathrm{CFL}}{\lambda_i^I + \lambda_i^V} \tag{3.6}$$

where CFL is the Courant-Friedrichs-Lewy number, and λ_i^I and λ_i^V are based on the spectral radius of the inviscid and viscous flux Jacobian matrices respectively at each point i

$$\begin{aligned}
\lambda_i^I &= \sum_{j=0}^{n_i}\left(\left|b_{(i)j-\frac{1}{2}}u_j + c_{(i)j-\frac{1}{2}}v_j + d_{(i)j-\frac{1}{2}}w_j\right| + a_i\sqrt{b^2_{(i)j-\frac{1}{2}} + c^2_{(i)j-\frac{1}{2}} + d^2_{(i)j-\frac{1}{2}}}\right) \\
\lambda_i^V &= \frac{\gamma}{Re}\sum_{j=0}^{n_i}\left(\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)_{ij}\frac{1}{\rho_{ij}}\left(b^2_{(i)j-\frac{1}{2}} + c^2_{(i)j-\frac{1}{2}} + d^2_{(i)j-\frac{1}{2}}\right)\right)
\end{aligned}$$

where a is the local speed of sound. The index ij indicates an averaged variable across the edge connecting points i and j.

To perform time accurate, unsteady calculations Eq. (3.1) is solved in real-time t,

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -\mathbf{R}^{n+1} \tag{3.7}$$

where the superscript n denotes the time level in real-time. For such calculations it is necessary to increase the order of accuracy in time, as is done in the dual-time stepping method [104]; so a second order $k = 2$ option in Eq. (3.4) is taken in real-time, so that Eq. (3.7) becomes

$$\frac{3\mathbf{w}^{n+1} - 4\mathbf{w}^n + \mathbf{w}^{n-1}}{2\Delta t} + \mathbf{R}^{n+1} = \mathbf{R}^* = 0 \tag{3.8}$$

where $\mathbf{R}^*$ is defined to be the unsteady residual vector, and Δt denotes a time step in real-time. This is a non-linear system of equations that cannot be solved directly; instead, we use the non-linear sub-iteration over $\mathbf{w}^{n+1}$ mentioned above, and view Eq. (3.8) as a modified pseudo-time steady state problem, which can be solved iteratively for $\mathbf{w}^{n+1}$. This is done by introducing a derivative with respect to a fictitious

pseudo-time for the unsteady residual vector

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\tau}^{m+1} = -\mathbf{R}^{*,m+1} \tag{3.9}$$

The steady state solution to Eq. (3.9) satisfies $\mathbf{R}^* = 0$, and so is also the solution of the unsteady system given by Eq. (3.8). In Eq. (3.9) the unsteady residual is evaluated at the new pseudo-time level $m+1$, and is therefore expressed in terms of the unknown solution at this new time level. To solve this problem, the same linearisation method in Eq. (3.3) is used in pseudo-time for the unsteady residual vector, which gives

$$\mathbf{R}^{*,m+1} \approx \mathbf{R}^{*,m} + \frac{\partial \mathbf{R}^*}{\partial \mathbf{p}} \Delta \mathbf{p}$$

Using this linearisation in Eq. (3.9), and from the definition of the unsteady residual

$$\frac{\partial \mathbf{R}^*}{\partial \mathbf{p}} = \frac{3}{2\Delta t} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}$$

we obtain the following system for the updates

$$\left(\left(\frac{1}{\Delta \tau} + \frac{3}{2\Delta t} \right) \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \frac{\partial \mathbf{R}}{\partial \mathbf{p}} \right) \Delta \mathbf{p} = - \left(\frac{3\mathbf{w}^m - 4\mathbf{w}^n + \mathbf{w}^{n-1}}{2\Delta t} + \mathbf{R}^m \right) \tag{3.10}$$

Thus, when the solution is converged in pseudo-time by iterating over m, we obtain $n+1$ and the real-time step Δt proceeds. The pseudo-time problem can be solved using any time marching method designed to solve steady state problems, utilising the standard acceleration techniques. The main advantage of the implicit method is that it allows the real-time step to be chosen for time accuracy alone, without worrying about numerical stability restrictions.

3.3 Choice of linear solution method

We can write Eqs. (3.5) for time-independent simulations, or (3.10) for time-dependent simulations, in a simpler global matrix form

$$A\Delta \mathbf{p} = -\mathbf{R} \tag{3.11}$$

where A represents the implicit system matrix, the construction of which is discussed in detail in Section 3.5. This equation encapsulates the implicit scheme, which is solved for $\Delta \mathbf{p}$ until $|\mathbf{R}| < \epsilon$, where ϵ is a small tolerance used to judge the convergence. The residual is normalised against the residual calculated on the first step.

The non-linear nature of the governing equations means that a direct method (such as Gaussian elimination) cannot produce a solution in a single iteration [105]; instead, Eq. (3.11) must be solved iteratively, requiring many solves for $\Delta \mathbf{p}$ until the solution

for Eq. (3.2) is obtained. The Newton-Raphson method is obtained in Eq. (3.5) if the exact Jacobian matrix of $\mathbf{R}$ is used, in conjunction with a direct solver, with a time step that is large enough so that is approaches infinity. Quadratic convergence can be achieved with this method, which results in a very fast asymptotic convergence to machine precision. Even though this gives the most efficient convergence rate in terms of the number of iterations, such a method is impractical for use in many CFD applications. Solving the linear system exactly at each iteration with a direct solver is extremely costly in terms of both memory requirements and arithmetic operations (which are often of the order $\mathcal{N}^3$, where $\mathcal{N}$ is the size of the system); there is also often much effort that must first be expended before the solution enters the region of quadratic convergence.

Instead, a more efficient method is obtained by using an inexact linear system solver, which only partially solves the system for $\Delta\mathbf{p}$ at each iteration. This means that great expense is spared in not driving the norm of the residual down to unnecessarily low values at each time step, so ϵ would be much greater than required for quadratic convergence. The inexact solution is then used in the calculation of the residual vector and the Jacobian matrix at the next time step; finite values of $\Delta\tau$ are used for additional stability. Despite losing the highly desirable quadratic convergence, such a method tends not to have the computational intensity associated with Newton-Raphson methods. It also means that the obtained solution of Eq. (3.2) does not have to be exact, the calculation of which is often unnecessary in most practical CFD calculations.

Not using an exact solver also means that there is little need to compute the Jacobian matrix exactly at each time step. Indeed, if $\mathbf{p}$ is the exact solution of the system, then $\mathbf{R} = 0$, and therefore $\Delta\mathbf{p} = 0$ independently of A, provided A is non singular; consequently, A may be chosen freely to improve the convergence, as it has no effect on the fully converged solution. The use of an approximate Jacobian matrix can make the formulation simpler; reduce memory requirements and operational counts; make the parallel implementation easier; and, most importantly, can improve the conditioning of the system. The ideal choice of A is such that it is relatively inexpensive to construct and solve the linear system for, but will still give an efficient convergence rate. Thus, we try to achieve the optimum balance between speed of convergence and cost of iterations.

3.4 Linear solver method

The inexact linear solver, used for each iteration of Eq. (3.11), is itself an iterative method, in which successive approximations are made to an initial solution vector using matrix-vector operations until the solution converges to a chosen tolerance. The iterations required to solve Eq. (3.11) are referred to as inner iterations, to distinguish them from the iterations of each solution of Eq. (3.11), which are used to obtain the solution of Eq. (3.2). The space which spans the approximations is called the Krylov

subspace; and for matrix-vector multiplications of the form $A\mathbf{v}$, the Krylov subspace of order m is defined as

$$\mathcal{K}_m(A, \mathbf{v}) = span\{\mathbf{v}, A\mathbf{v}, \ldots, A^{m-1}\mathbf{v}\}$$

For most Krylov subspace methods, the exact solution can be obtained after at most $\mathcal{N}$ steps. However, if $\mathcal{N}$ is very large it is not practical to store the whole Krylov subspace; thus, the use of such procedures as direct methods is limited. Instead, a good approximation to the solution in far fewer steps is found; so when the iteration reaches a cut-off step, the subspace is deleted and the latest solution vector is used as the initial solution for the restarted procedure. In this way the iteration method is *itself* applied iteratively.

The Krylov subspace generally only contains a good approximate solution, for any moderate size m, if the system is well-conditioned. If this is not the case, then one can modify the original problem to obtain a better Krylov subspace. This can be done using a preconditioner matrix C. In this work a left preconditioning is used, which results in solving the modified problem

$$C^{-1}A\Delta\mathbf{p} = -C^{-1}\mathbf{R}$$

The preconditioner is designed so that $C^{-1}A$ in some sense approximates the identity matrix; so $C^{-1}(\mathbf{R} - A\mathbf{x}_k)$ can be expected to approximate the error in an approximate solution $\mathbf{x}_k$. The next approximate solution $\mathbf{x}_{k+1}$ can then naturally be obtained by taking

$$\mathbf{x}_{k+1} = \mathbf{x}_k + C^{-1}(\mathbf{R} - A\mathbf{x}_k) \tag{3.12}$$

It is necessary to provide criteria to make it likely for the correct solution for $\Delta\mathbf{p}$ to be approached with increasing k. To do this, a modified version of Eq. (3.12) is used, with additional parameters introduced into the iteration so that the successive approximations to the solution vector are formed by minimising the norm of the residual vector. It is called the generalised conjugate residual (GCR) method [106]; and in this method the matrix A is not required to be symmetric and positive-definite, making it advantageous in that the method depends on A, rather than the normal equations formed by A.

Various iterative algorithms have been tested for finite volume codes [107], and it was concluded that the choice of method is not as crucial as the preconditioning. Obviously the most effective preconditioner would be such that $C^{-1}A = I$ exactly, but this would be too expensive to compute. Instead, the preconditioning strategy is based on a Block Incomplete Lower-Upper (BILU) factorisation [108]. As with the Jacobian matrix, the blocks refer to the points in the domain, and are square blocks with a size of the number of primitive variable unknowns. An approximate factorisation of

the matrix A into lower and upper factors is calculated, with the sparsity pattern of the factorisation restricted according to an algorithm that allows the calculation of progressively better approximations to the inverse, at the cost of including more terms (levels) in the factors. The levels of fill-in are given integer values, level l or BILU(l); however, to keep the operation as efficient as possible, we use the method with zero fill-in, BILU(0), which means that the sparsity pattern on the lower and upper matrices is the same as the one in the Jacobian matrix.

The calculation of the preconditioner at each implicit time step takes a non-trivial amount of time. If the linear system in Eq. (3.11) is well-conditioned, then it can be solved in very few (often about five) steps, and the calculation of the preconditioner can take anything up to 50% of the linear solution time. As a result, using the same preconditioner, and only recalculating it after a certain number of linear systems have been solved (say ten implicit solution steps), makes a large saving in computational time.

3.5 Constructing the approximate Jacobian matrix

To describe the approximate Jacobian matrix, it is helpful to describe the form of the exact Jacobian matrix as a starting point, and then the simplifications that follow. If we first consider the structure of the Jacobian matrix, it has local contributions consisting of the partial derivatives of the residual vector at a point i, with respect to the primitive variables at every point j, within the domain Ω. In principle $\mathbf{R}$ may be a function of $\mathbf{p}_j \forall j \in \Omega$, so the global Jacobian matrix can be written as an $N \times N$ block matrix, where N is the number of points in the computational domain; each of these blocks (denoted $\frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j}$) consists of an $l \times l$ matrix, where l is the number of unknowns in $\mathbf{p}$. For each point, the terms in each block matrix can be found by differentiating the right hand side of Eq. (2.29), so in two-dimensions

$$\begin{aligned}
\frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} &= \frac{\partial}{\partial \mathbf{p}_j} \left\{ \sum_{k=0}^{n_i} \left(b_{ik-\frac{1}{2}} (\mathbf{f}_{k-\frac{1}{2}} - \mathbf{f}^v_{ik}) + c_{ik-\frac{1}{2}} (\mathbf{g}_{k-\frac{1}{2}} - \mathbf{g}^v_{ik}) \right) + \mathbf{S}_i \right\} \\
&= \sum_{k=0}^{n_i} \left\{ b_{ik-\frac{1}{2}} \left(\frac{\partial \mathbf{f}_{k-\frac{1}{2}}}{\partial \mathbf{p}_j} - \frac{\partial \mathbf{f}^v_{ik}}{\partial \mathbf{p}_j} \right) + c_{ik-\frac{1}{2}} \left(\frac{\partial \mathbf{g}_{k-\frac{1}{2}}}{\partial \mathbf{p}_j} - \frac{\partial \mathbf{g}^v_{ik}}{\partial \mathbf{p}_j} \right) \right\} + \frac{\partial \mathbf{S}_i}{\partial \mathbf{p}_j}
\end{aligned} \tag{3.13}$$

The final statement can be made since the shape functions are found purely from geometric considerations, and are not dependent on the flow variables. From Eq. (3.13) we can see that the flux derivatives with respect to the primitive variables are needed; these derivatives are obtained analytically. The inviscid fluxes are formed by solving the Riemann problem, as outlined in Section 2.3.1, so when differentiated there will be left and right side block matrix contributions to the Jacobian matrix, corresponding to

the Riemann problem solved. Similarly, the viscous fluxes and turbulence model use local flow variables and gradients at both the left and right states.

It is clear that $\mathbf{R}_i$ in Eq. (3.13) depends only on the points that belong to $\mathcal{M}(i)$ that are used to form the inviscid and viscous fluxes. This means that most of the elements of the Jacobian matrix are zero, resulting in a sparse matrix. For a first order scheme the inviscid flux is dependent only on the $\mathcal{M}(i) = n_i$ points that make up the meshless stencil. From Eq. (2.22), we can see that there will, therefore, be two contributions to the Jacobian matrix: one for the left state, which is for the star point on the diagonal (since $j = i$); and one for the right, which is for the neighbouring point ($j \neq i$). These contributions can be written

$$A_L = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_L} \qquad A_R = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_R} \tag{3.14}$$

where these matrix blocks are added to the global Jacobian matrix in the same way as the fluxes are for the residual vector $\mathbf{R}$.

The second order scheme has reconstructed left and right states of the Riemann problem in Eq. (2.23), which are each dependent on the gradients of the left and right side variables as well. As a result, there are additional non-zero contributions to the Jacobian matrix, as $\mathcal{M}(i) > n_i$, corresponding to each point within the neighbouring stencils that form the least squares gradients for the higher order reconstruction. The chain rule is used to provide the corresponding terms in the matrix. For each Riemann problem, there are several matrix blocks for each of the left and right states, written as

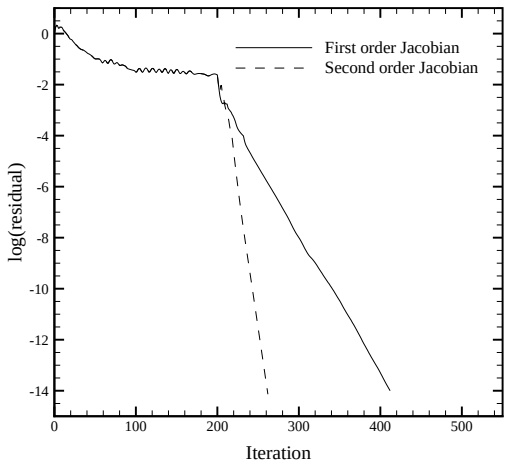
$$A_L = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_L}\frac{\partial \mathbf{p}_L}{\partial \mathbf{p}_j} \qquad A_R = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_R}\frac{\partial \mathbf{p}_R}{\partial \mathbf{p}_j} \tag{3.15}$$

The additional terms compared to Eq. (3.14) are the cause of the greater fill-in for the second order Jacobian matrix.

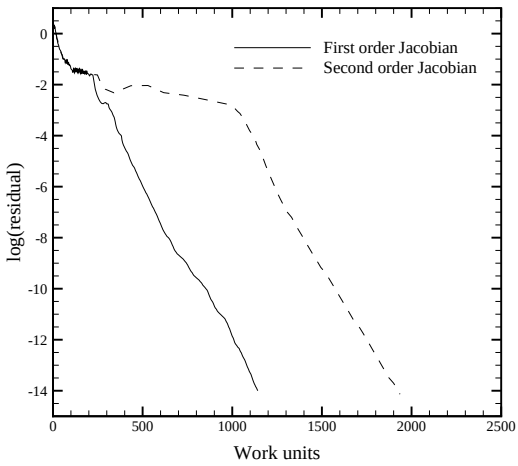
The larger number of non-zero terms in the second order Jacobian matrix can cause some difficulties for the linear solver method: firstly, the additional non-zeros mean that it is more costly to calculate; secondly, the memory requirements are greater due to the increased storage; thirdly, each inner-iteration is slower to perform since the matrix-vector multiplications of the iterative solver require more operation counts; and finally (and most importantly), since the additional terms are off-diagonal they can increase the stiffness and ill-conditioning of the matrix, making it much more difficult to solve the linear system efficiently.

To avoid this extra cost, we keep the number of non-zero entries in the Jacobian matrix as low as possible by omitting the second order block contributions to the Jacobian matrix. In other words we enforce

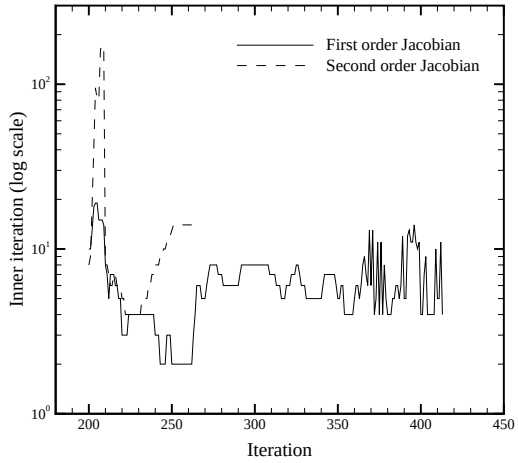
$$\frac{\partial \mathbf{p}_L}{\partial \mathbf{p}_j} = \delta_{jL} I \qquad \frac{\partial \mathbf{p}_R}{\partial \mathbf{p}_j} = \delta_{jR} I$$



(a) Convergence of the non-linear system in Eq. (3.2) in terms of iterations



(b) Convergence of the non-linear system in Eq. (3.2) in terms of CPU time



(c) Number of inner-iterations used to solve Eq. (3.5), per iteration to solve Eq. (3.2)

Figure 3.1: Comparison of convergence rates using first and second order Jacobian matrices for the inviscid, transonic NACA 0012 case in Section 4.1.1.

in Eq. (3.15), where δ is the Kronecker delta symbol and I is the identity matrix. This means that the second order residual vector $\mathbf{R}_i$ is used in the calculation of the blocks, but the sparsity pattern of the global matrix is the same as that of the first order Jacobian matrix.

The benefit of this approach can be seen in Fig. 3.1, which compares the rate of convergence when using the inexact linear solver with first and second order Jacobian matrices. The second order Jacobian in this case is not exact though, as the flux limiter is not differentiable. The plots are for the two-dimensional, inviscid, transonic, steady state case outlined in Section 4.1.1. Implicit schemes require particular treatment during the early stages of the iterative procedure; the usual approach in starting the method is to take a small implicit time step as a start-up, and to increase it later on. It was found, however, that smoothing out the initial flow by doing some explicit Euler

iterations of the form

$$\mathbf{w}^{m+1} = \mathbf{w}^m - \Delta\tau\mathbf{R}^m,$$

and then switching to the implicit scheme was equally efficient; hence, two hundred explicit iterations were performed first before the implicit scheme starts. Figure 3.1(a) shows that the use of a second order Jacobian matrix to solve the non-linear system in Eq. (3.2) is more efficient in terms of the number of iterations that it takes for convergence. However, Fig. 3.1(b) shows the increased cost in CPU time of using a second order Jacobian matrix; a work unit is defined as the amount of time required for one explicit time step. The use of a second order Jacobian matrix incurs about twice the cost of the first order Jacobian matrix because of the difficulties which were outlined earlier. These difficulties can be seen further in Fig. 3.1(c), which shows the number of inner iterations that are required to solve the linear system in Eq. (3.5) at each step. The convergence of the linear solver is also determined by using the $L2$ norm of the residual, which has been normalised against the residual calculated on the first step. In this work, the level at which convergence of the linear solver is achieved is at $\epsilon = 10^{-3}$. It can be seen that although fewer iterations are required using the second order Jacobian matrix (which is in agreement with Fig. 3.1), many more inner iterations must be performed to achieve this level of convergence; this is particularly so during the first few iterations (note the log scale) of the implicit scheme, which can also be seen in Fig. 3.1(b) when over 750 work units are expended for very little resultant convergence. This demonstrates the poor conditioning of the second order Jacobian matrix, which is the major problem with its use.

The Jacobian matrices obtained from the viscous fluxes in Eq. (3.13) are formed in a similar manner. The compact support of the viscous stencil for a point is the same as that for the inviscid, second order Jacobian matrix. Once again, the approximate Jacobian matrix is formed by omitting the terms that would increase the fill-in of the exact first order inviscid Jacobian matrix. For the RANS equations with the Spalart-Allmaras turbulence model, the governing equations are solved fully coupled with the turbulence model equations (though the option of decoupling the turbulent, and mean flow terms is available), so the vector of primitive variables becomes $\mathbf{p} = (\rho, u, v, w, p, \tilde{\nu})^T$. This coupling of the mean flow and turbulent equations means that the matrix blocks are larger; and the same approximations are made to the global Jacobian matrix for the terms corresponding to the convective and diffusive fluxes in the Spalart-Allmaras transport equation. There is also the additional Jacobian matrix contribution of the source term to be considered. Currently, this Jacobian matrix is exact, as the contributions are limited to the set $\mathcal{M}(i)$ for each point anyway. Constructing the approximate Jacobian matrices in this way means that the sparsity pattern of the inviscid, laminar and turbulent approximate Jacobian matrices are the same.

The Jacobian matrix must also contain contributions that arise from the boundary conditions. Recall from Section 2.3.4 that the boundary conditions are imposed on the star point by the halo points, which have flow variables obtained from their corresponding interior points. The halo points are not updated, as they do not belong to the domain Ω, and as there are no terms in the residual vector for the halo points, there will not be any additional rows or columns in the Jacobian matrix. Only the boundary points have halos, so for the star point i on the boundary, we can again use the chain rule to calculate the contributions due to the halo points on the right side of the Riemann problem, so

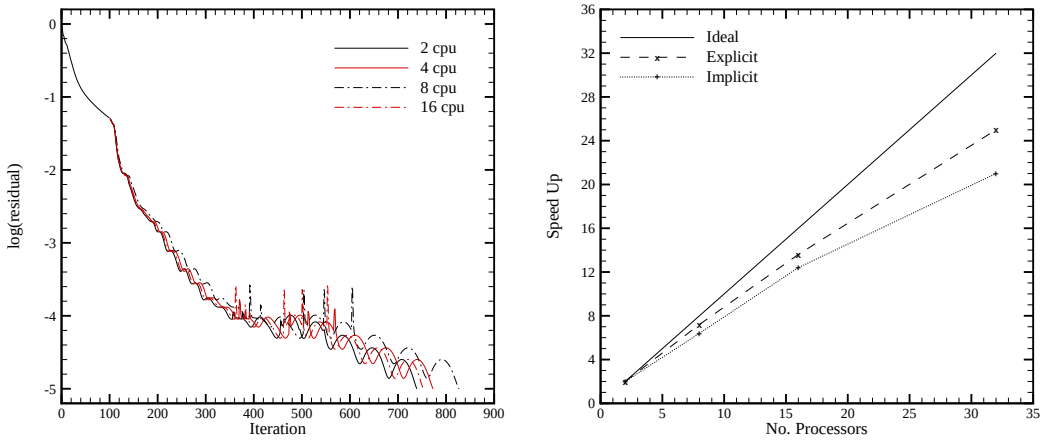
$$A_R = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_h} \frac{\partial \mathbf{p}_h}{\partial \mathbf{p}_j} \tag{3.16}$$

The contribution A_R from the halo point is expressed in terms of the internal points, which can then be added into the Jacobian matrix in the correct place. As the matrix $\frac{\partial \mathbf{p}_h}{\partial \mathbf{p}_j}$ is non-zero only when j is a point within the stencil Ω_i, the sparsity pattern of the Jacobian matrix is not altered by the contributions from the boundary points.

3.6 Parallel implementation

Although single processors are becoming increasingly powerful, they are not yet sufficient to perform the large scale computations that are needed in industrial applications. The limits of single processor performance mean that it is necessary for a CFD code to be able to perform calculations on multiple processors. This is done, in this work, with a distributed memory parallelism, where each processor has access to part of the data which resides in the memory it is directly connected to [88]; thus, each processor behaves almost as if it is a separate serial program, responsible for solving the equations at a distinct part of the computational domain. The domain decomposition is achieved using a library called METIS [109].

There is no data shared between processors; access to data which resides in the memory attached to another processor is done through the network via message passing using the MPI (message passing interface) library. The data consists of the points used in the meshless discretisation, and is split across the processors according to the number of processors available. For an efficient parallelisation each processor should have roughly the same amount of work, so will have the same number of points, while keeping the communication to a minimum. Communication is necessary because points that form stencils along the boundaries of the domain decomposition may belong to different processors; as a result, for each processor, a list of communication points is created, which contains points that are not stored in the current processor, but are included in the local stencils. The data that is communicated between processors for a communication point (denoted j) consists of: the local flow variables ϕ_j, the local



(a) Parallel convergence histories using an increased number of processors

(b) Parallel speed-up (per iteration) as a function of number of processors

Figure 3.2: Parallel solver performance for the M6 wing test case at inviscid flow conditions analysed in Section 4.2.3.

gradients $\nabla\phi_j$, the point coordinates $\mathbf{x}_j$, and the flux limiter values ψ_j; all are used in Eq. (2.23) for the solution of the Euler equations.

When using the implicit scheme, the Jacobian matrix is also divided among the processors in the same way; consequently, each processor requires the matrix information for its local and communication points. The use of an approximate Jacobian matrix reduces parallel communication since there are fewer communication stencil points; it also means that the meshless shape functions do not need to be communicated, which would be necessary to form the additional terms in Eq. (3.15) for the second order Jacobian matrix.

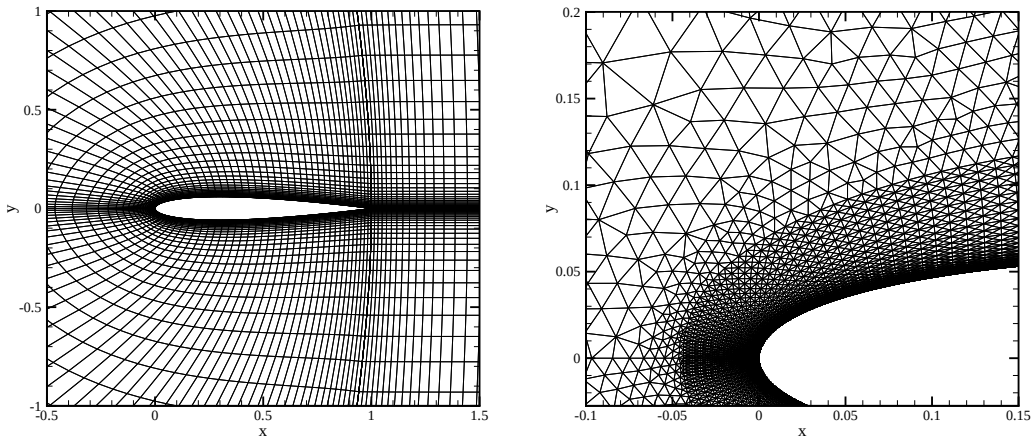
The parallel communication can be further minimised by decoupling the BILU(0) factorisation between the blocks in the Jacobian matrix; hence, there is no parallel communication when forming the preconditioner matrix used in Eq. (3.12). This approach improves the parallel performance, but at the expense of not forming the best BILU(0) possible. This has an effect on the convergence rate of the linear solver, as can be seen in Fig. 3.2(a), which shows the convergence in terms of the number of iterations for the solution of the Euler equations on an M6 wing on multiple processors; the flow results for this test case are given in Section 4.2.3 in the next chapter. Despite this, the speed-up using an increasing number of processors can be seen in Fig. 3.2(b). A linear rate can, realistically, never be reached due to the parallelisation overheads introduced in the domain decomposition, namely, the communication and load balancing.

Chapter 4

Method Evaluation

4.1 Two-dimensional cases

Results are presented in this chapter for validation purposes and to build confidence in the developed meshless flow solver, which is applied to the multibody systems analysed in the following chapters. The two-dimensional solutions presented include standard steady and unsteady aerofoil test cases, for inviscid and viscous flows, using the NACA 0012 aerofoil. Solution convergence studies are made using successively finer point distributions, and comparisons are made with the parallel multiblock (PMB) flow solver [110], which is an established finite volume research code. Also included are circular cylinder cases at laminar flow conditions that demonstrate the method outlined in Section 2.3.2. These results are compared with experimental data and numerical solutions found in the literature where available. RANS results, using the formulation of Section 2.3.3, for steady and unsteady test cases are also presented, with comparisons with PMB and experimental data for further validation.



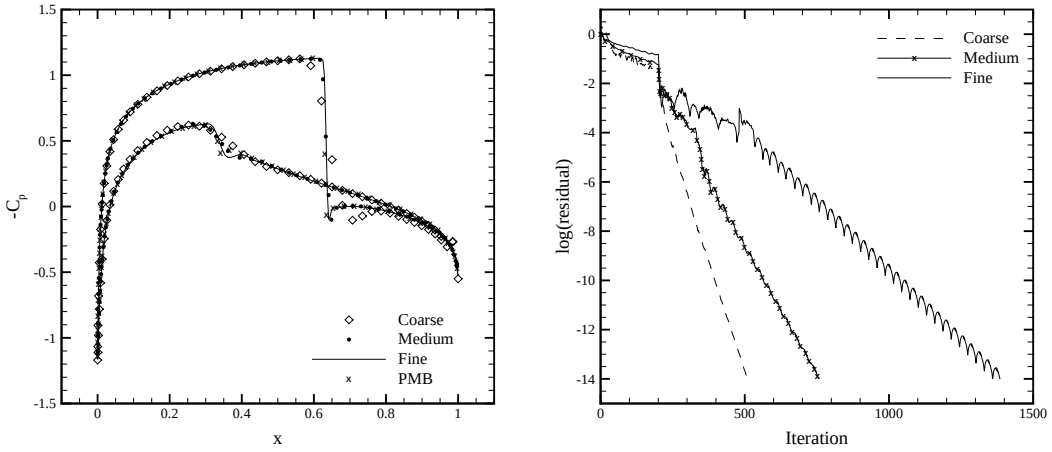
(a) Block structured grid

(b) RANS connectivity used by PML

Figure 4.1: NACA 0012 aerofoil connectivities.

	PMB			PML linear			PML quadratic		
	C_l	C_d	C_m	C_l	C_d	C_m	C_l	C_d	C_m
C	0.339	0.0230	-0.034	0.327	0.0183	-0.032	0.332	0.0203	-0.035
M	0.341	0.0220	-0.035	0.341	0.0201	-0.036	0.344	0.0212	-0.035
F	0.343	0.0218	-0.035	0.344	0.0211	-0.037	0.345	0.0216	-0.035

Table 4.1: Convergence of lift, drag and moment coefficients for inviscid transonic case using linear and quadratic reconstructions using coarse (C), medium (M) and fine (F) point distributions.



(a) Surface pressure distribution

(b) Convergence history

Figure 4.2: Transonic inviscid flow for NACA 0012 aerofoil at $M_\infty = 0.8$ and α=1.25° using a quadratic polynomial reconstruction.

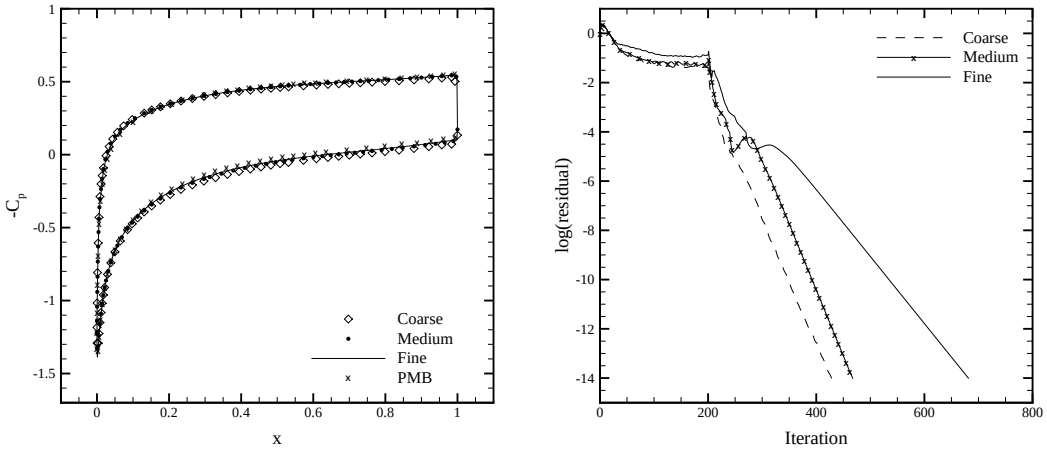
4.1.1 Steady inviscid flow over a NACA 0012 aerofoil

In this subsection the Euler equations are solved at transonic and supersonic flow conditions. A point distribution study is made using points obtained from structured, multiblock grids consisting of 5686, 22380 and 88792 points, regarded as coarse, medium and fine respectively; there are 128 points on the solid wall for the coarse, 256 for the medium and 512 for the fine point distributions. The topology of the grids can be seen in Fig. 4.1(a); and the minimum point spacing normal to the wall is $10^{-4}c$ where c is the chord length. The PMB calculations are made using these finite volume grids; while the meshless stencils used by PML use the connectivity of the grids: consisting of the nine points that make up the four cells surrounding a star point for an interior stencil; and the six points that make up the two cells surrounding a star point for a boundary stencil.

The results presented include the surface pressure distribution plots for each case, along with the lift (C_l), drag (C_d) and moment coefficient about the quarter chord (C_m). The coefficients are presented for the cases of using a linear ($m = 4$) and a quadratic

	PMB			PML linear			PML quadratic		
	C_l	C_d	C_m	C_l	C_d	C_m	C_l	C_d	C_m
C	0.521	0.1551	-0.111	0.529	0.1507	-0.114	0.530	0.1507	-0.113
M	0.523	0.1545	-0.111	0.527	0.1528	-0.113	0.526	0.1531	-0.112
F	0.521	0.1542	-0.111	0.526	0.1524	-0.113	0.525	0.1538	-0.112

Table 4.2: Convergence of lift, drag and moment coefficients for inviscid supersonic case using linear and quadratic reconstructions using coarse (C), medium (M) and fine (F) point distributions.



(a) Surface pressure distribution

(b) Convergence history

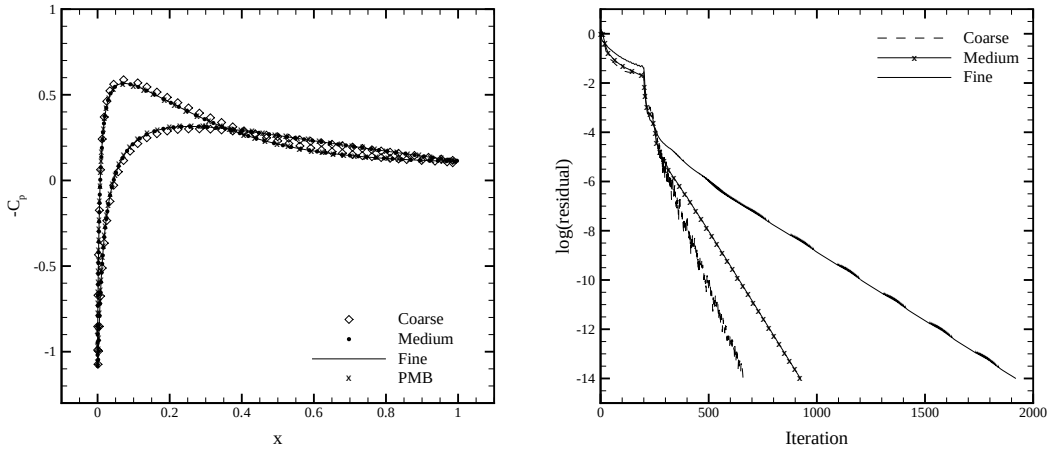
Figure 4.3: Supersonic inviscid flow for NACA 0012 aerofoil at $M_\infty = 1.2$ and $\alpha = 7.0^\circ$ using a quadratic polynomial reconstruction.

($m = 6$) reconstruction in Eq. (2.7); and can be compared with the PMB solutions to determine the accuracy and solution convergence of the scheme. The convergence histories for each computation are also presented: the level of convergence in pseudo-time for the steady state simulations is determined by the norm of the residual, which is normalised with the residual calculated after the first pseudo-time step. For each test case the initial flow field is first smoothed out by 200 explicit iterations, after which the implicit scheme is used. The explicit CFL number is chosen to be 1 for all cases, while the choice of implicit CFL number is as decided below.

The first case is a transonic flow with freestream Mach number (M_∞) of 0.8 and an angle of attack (α) of 1.25°. The lift, drag and moment coefficients are displayed in Table 4.1 for the linear and quadratic reconstructions. It can be seen that, compared with the linear reconstruction, the results from the quadratic reconstruction converge to a level that is in better agreement with the values given by PMB. Using a linear reconstruction also means that a smaller CFL number must be used to achieve convergence. For this case the CFL number is 30, while for the quadratic reconstruction a larger CFL number can be used. This may be because the higher order polynomial is

	PMB			PML linear			PML quadratic		
	C_l	C_d	C_m	C_l	C_d	C_m	C_l	C_d	C_m
C	0.088	0.0635	0.022	0.088	0.0710	0.021	0.089	0.0594	0.021
M	0.061	0.0592	0.023	0.075	0.0631	0.021	0.068	0.0546	0.022
F	0.055	0.0583	0.023	0.062	0.0612	0.022	0.059	0.0562	0.023

Table 4.3: Convergence of lift, drag and moment coefficients for laminar case using linear and quadratic reconstructions using coarse (C), medium (M) and fine (F) point distributions.



(a) Surface pressure distribution

(b) Convergence history

Figure 4.4: Laminar flow for NACA 0012 aerofoil at M_∞ = 0.5, α=3.0° and Re=5000 using a quadratic polynomial reconstruction.

a better approximation of the flow variables that are approximated, using the meshless scheme, leading to a more stable system to be solved. For the convergence plot in Fig. 4.2(b), the quadratic reconstruction is used with an implicit CFL number of 80. It can be seen that, using the implicit scheme, convergence is achieved for all point distributions. In doing this, there is, on average, six iterations per linear solve at each pseudo-time step (as can also be seen in Fig. 3.1(c)). The transonic shocks can be seen in Fig. 4.2(a), and are well resolved for the medium and fine point distributions.

The supersonic case ($M_\infty = 1.2$, $\alpha = 7.0°$) has coefficients presented in Table 4.2, which show that the results are converging to a value similar to those of PMB. The same CFL numbers, of 30 for the linear case, and 80 for the quadratic case, were used for these computations. The surface pressure coefficient plot is given in Fig. 4.3(a), which shows good agreement with PMB. The convergence rate in Fig. 4.3(b) is good for this case too; though a lot of work is done, at the 200th iteration, in switching from the explicit to the implicit time scheme, as over 100 inner iterations are needed for the linear solve at this step. The convergence for the subsequent iterations is more efficient as, on average, only five inner iterations are required per pseudo-time step.

Solution	Angle of separation θ_s (.°)	C_d
PML (coarse)	127.43	1.58
PML (medium)	126.64	1.55
PML (fine)	126.23	1.54
Ref. [111]	125.8-127.3	1.48-1.62

Table 4.4: Angle of separation and drag coefficients for laminar flow over circular cylinder with Re=40.

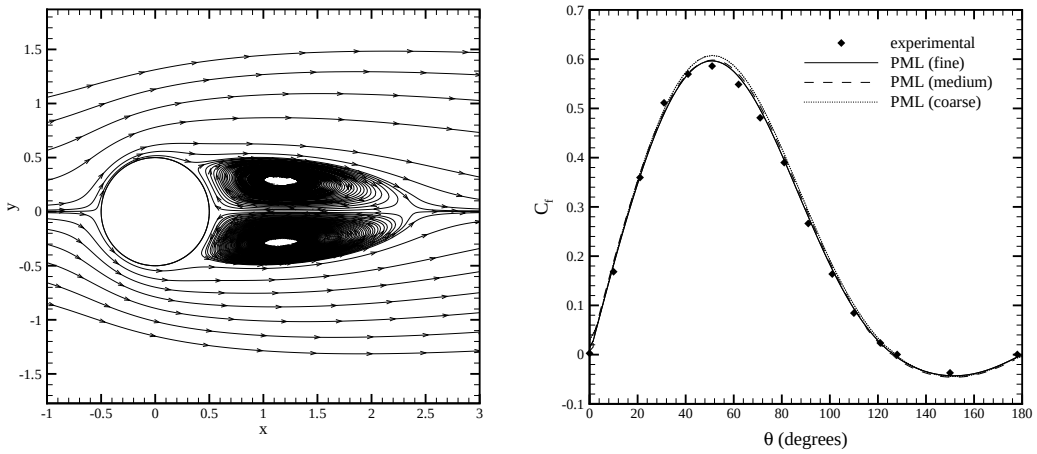
4.1.2 Steady laminar flow over a NACA 0012 aerofoil

The same point distributions are used for a study at the laminar flow conditions of $M_\infty = 0.5$, $\alpha = 3^\circ$ and Reynolds number (Re) of 5000. The lift, drag and moment coefficients for the linear and quadratic reconstructions are presented in Table 4.3. The coefficients can be seen to be converging towards the solution provided by PMB; though the difference with the linear approximation is much greater than for the quadratic approximation because the function in Eq. (2.7) must approximate the viscous fluxes in addition to the inviscid flux. For this reason all subsequent viscous calculations are made using the quadratic reconstruction in Eq. (2.7).

The surface pressure distributions in Fig. 4.4(a) show good agreement between PMB and the point distributions used by PML. The rate of convergence, shown in Fig. 4.4(b), is fast for the coarse and medium point distributions; though it is not as efficient with the fine case after convergence of four orders of magnitude has been reached. The number of iterations remains at about eight per linear solve all through the computation: this increase in iteration count is due to the further approximations to the Jacobian that are used for the viscous fluxes.

4.1.3 Steady laminar flow over a circular cylinder

To further demonstrate the ability of the flow solver for the simulation of laminar flows, circular cylinder steady and unsteady cases are presented. For the steady case, three cylinder point distributions are used to determine the solution convergence. These are labelled coarse, with 7920 points, 120 of which are on the solid wall; medium, with 31440 points, of which 240 on the solid wall; and fine, with 52350 points, with 480 on the solid wall. The far field is located at 40 diameters from the cylinder centre. The Reynolds number is chosen to be $Re = 40$ for this simulation: this test case is characterised by the presence of a wake bubble behind the cylinder, which can be seen in the streamline plot given by PML in Fig. 4.5(a). The angle of separation, with $\theta_s = 0^\circ$ corresponding to the front stagnation point, and the drag coefficients are given in Table 4.4. The skin friction distribution is presented in Fig. 4.5(b), where the skin friction coefficient is normalised by reference freestream values. The experimental results are taken from Ref. [111] for comparison, and the results given by PML agree well.



(a) Velocity streamlines around the cylinder

(b) Skin friction distributions

Figure 4.5: Laminar flow over circular cylinder with Re=40.

4.1.4 Unsteady laminar flow over a circular cylinder

When the Reynolds number of the case in Section 4.1.3 is increased, then the wake becomes unsteady. It is one of many viscous flows which evolve into unsteady motions because of flow instability despite the fixed boundaries. The flow separation from the rear surface forms a broad pulsating wake, with vortices shedding alternately from the upper and lower parts of the rear surface. This effect, called the von Kármán vortex street, can be seen in Fig. 4.6, which is a contour plot of the x component of velocity, as computed by PML, for the case described.

A time-dependent simulation at $Re = 100$ is performed using coarse, medium and fine point distributions of 12484 points, with 114 making up the solid wall boundary; 25264 points, with 214 on the solid wall; and 99298 points, with 426 solid wall points, respectively. The far field of these domains extends to 100 diameter lengths from the cylinder centre. The real-time step was chosen to be 0.05 non-dimensional time units; though additional results are presented using a larger time step of $\Delta t = 0.25$ and a smaller time step of $\Delta t = 0.01$ for the medium point distribution. The simulations used a CFL number of 200; and, on average, there were ten pseudo-time steps per real-time step, each costing 35 work units, with the convergence in pseudo-time set to 10^{-3}. The convergence in pseudo time for time accurate simulations is determined by the $L2$ norm of the update in pseudo-time ($\Delta \mathbf{p} = \mathbf{p}^{m+1} - \mathbf{p}^{m}$), which is normalised with the $L2$ norm of the difference between flow variables in real-time ($\Delta \mathbf{p} = \mathbf{p}^{n+1} - \mathbf{p}^{n}$). The linear solver required on average four iterations per pseudo-time step.

The dimensionless shedding frequency, or Strouhal number, is presented in Table 4.5, with experimental values taken from Ref. [113]. Also presented are the mean pressure drag C_{dp} and mean friction drag C_{df} for each case. These values can be compared with those from Ref. [112], which are CFD results calculated using another meshless solver.

	Strouhal number	C_{dp}	C_{df}
PML Coarse ($\Delta t = 0.05$)	0.1266	0.8388	0.3383
PML Medium ($\Delta t = 0.25$)	0.1510	0.9302	0.3682
PML Medium ($\Delta t = 0.05$)	0.1515	0.9333	0.3699
PML Medium ($\Delta t = 0.01$)	0.1517	0.9335	0.3704
PML Fine ($\Delta t = 0.05$)	0.1613	0.9786	0.3438
Ref. [112]	-	0.9557	0.3236
Ref. [113]	0.16-0.17	-	-

Table 4.5: Comparisons of computed Strouhal number, mean pressure and mean friction drag coefficients with experimental data from Ref. [113] and numerical data from Ref. [112].

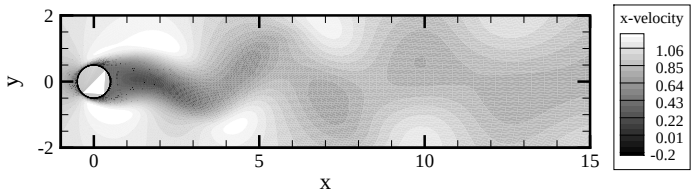
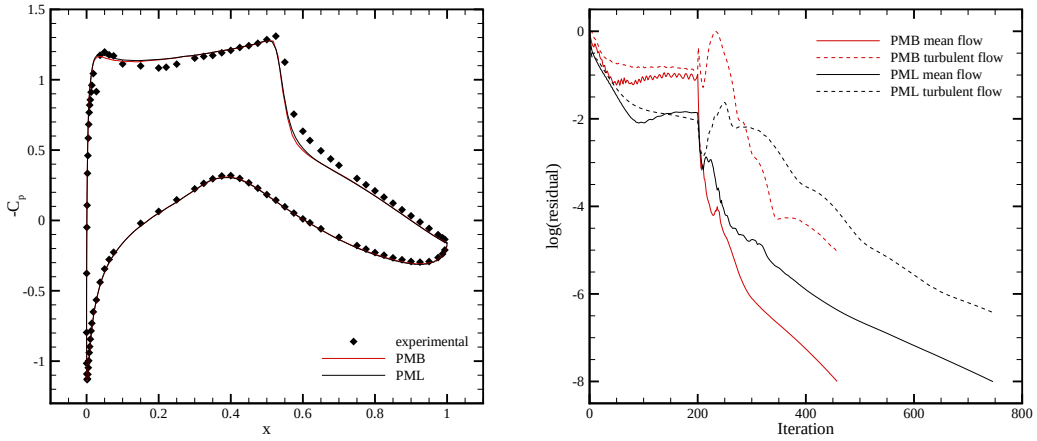


Figure 4.6: Instantaneous contour plot of the x component of velocity for the circular cylinder, unsteady case at *Re*=100.

4.1.5 Steady turbulent flow over the RAE 2822 aerofoil

To demonstrate the solution of the RANS equations, the well known steady state test case of transonic turbulent flow over an RAE 2822 aerofoil is presented. The flow conditions for this case, also known as case 9 from Ref. [114], are $M_\infty = 0.73$, $\alpha = 2.79°$ and $Re = 6.5$ million. The results are compared with the experimental data from the reference and PMB, which uses a block structured grid of 37752 points, with 348 points making up the solid wall. PML uses a different point distribution, consisting of highly stretched stencils, in a structured format, until a distance of about 10% of the chord from the boundary wall, while the rest of the domain has an unstructured grid format. The format is similar to the point distribution shown in Fig. 4.1(b); thus, the stencils in the unstructured region will contain six or seven points. The number of points in this domain is 35474, with 358 points on the solid wall. The domains used by PML and PMB both have a minimum wall normal spacing of 6×10^{-6}c.

On average there were twelve iterations per linear solve, and the simulation cost 2763 work units in total. The surface pressure distribution is given in Fig. 4.7(a), and there is good agreement between PML and the experimental data, and negligible difference with the PMB results. A convergence history of both flow solvers is presented in Fig. 4.7(b), using the maximum possible CFL number for each flow solver. For PMB this is 200; for PML it was slightly larger at 250. The convergence of the mean equations and Spalart-Allmaras turbulence model are given separately.



(a) Surface pressure distribution

(b) Convergence history

Figure 4.7: Pressure distributions and convergence histories for case 9 of Ref. [114].

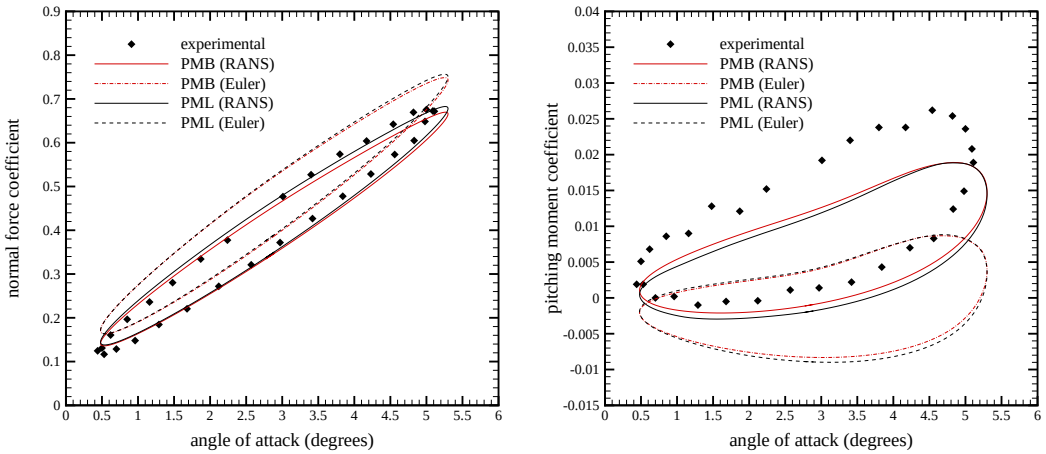
4.1.6 Unsteady turbulent flow - AGARD CT1

This test case concerns the simulation of a single NACA 0012 aerofoil undergoing a forced periodic motion, around the quarter chord of the aerofoil, prescribed by a sinusoidal function

$$\alpha(t) = \alpha_0 + \alpha_a \sin(2kt)$$

The configuration of the AGARD CT1 test case [115] exhibits intermittent shock motion during the cycle at freestream Mach number 0.6, with a reduced frequency of $k = 0.0808$, a mean incidence of $\alpha_0 = 2.89°$ and a pitch amplitude of $\alpha_a = 2.41°$. The Reynolds number is 4.8 million and the flow is assumed to be fully turbulent. Results are compared with the experimental data of the CT1 test case, obtained from the reference, and flow simulations using PMB. A multiblock grid of 30488 points with 264 points on the solid wall is used by PMB, while PML uses a point distribution, shown in Fig. 4.1(b), of 31574 points, with 348 on the solid wall. The minimum wall normal point spacing is 10^{-5}c for both domains.

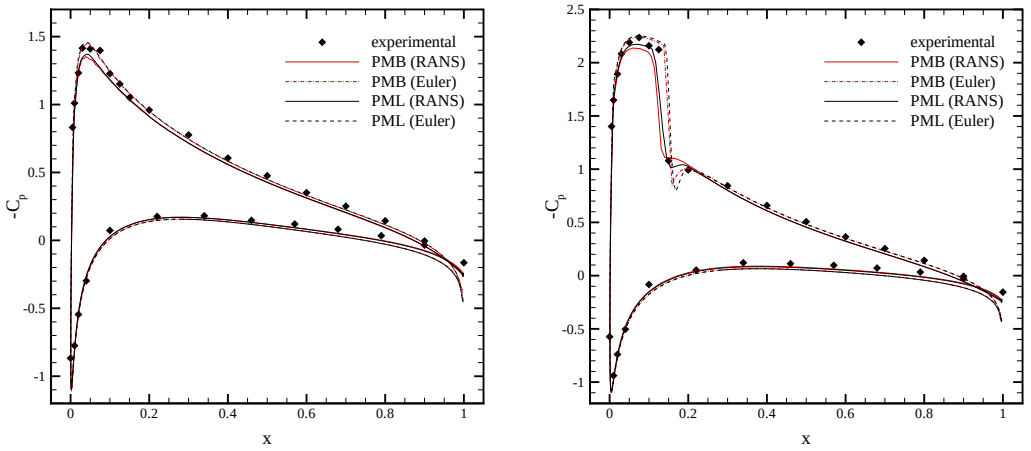
Five motion cycles, each consisting of 128 real-time steps, were simulated to solve both the RANS and Euler equations. For the turbulent simulation each of the real-time steps cost 426 work units on average, and required 184 pseudo-time steps to achieve convergence to 10^{-3}. Typically, three iterations of the linear solver were required per pseudo-time step with a CFL number of 60. Normal force and pitching moment coefficients are shown in Figs. 4.8(a) and 4.8(b) respectively, in which the overall agreement between PML and PMB can be determined. To explain the relatively large difference for the pitching moment coefficient between the RANS results and the experimental data, published results in Ref. [116] showed that the pitching moment value is sensitive to the moment centre used.



(a) Normal force coefficient (b) Pitching moment coefficient

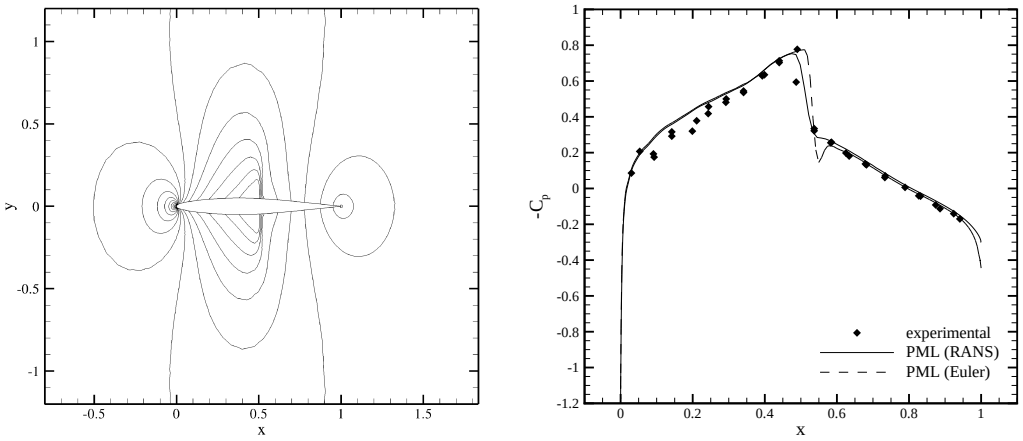
Figure 4.8: Normal force and pitching moment coefficients for forced pitching motion test case of AGARD CT1, comparisons between PML, PMB and experiment.

The inviscid results show a consistent trend compared with viscous simulation results and experiments; however, for the highest angles of attack during one cycle of motion, the difference between the flow models is more distinct. An explanation for this is due to the slight differences of the resolved shock wave, in both location and strength, formed during parts of the cycle of motion. This can be seen in Fig. 4.9, which presents unsteady pressure distributions at two instantaneous angles of attack during the cycle for both the inviscid and viscous calculations. The comparison between PMB and PML is good; and the difference in the shock wave predicted by the flow models can be seen clearly in Fig. 4.9(b) during the high angle of attack downstroke.



(a) $\alpha = 3.84°$ (upstroke) (b) $\alpha = 4.82°$ (downstroke)

Figure 4.9: Unsteady pressure distributions for forced pitching motion test case of AGARD CT1; comparisons of inviscid and viscous flow between PML, PMB and experiment at two angles of attack during upstroke and downstroke.



(a) RANS steady pressure contours

(b) Steady surface pressure distribution

Figure 4.10: Pressure contours and surface pressure distribution for M_∞=0.8, α=0° and Re=12.5 million, comparing experimental data and PML RANS and Euler computations.

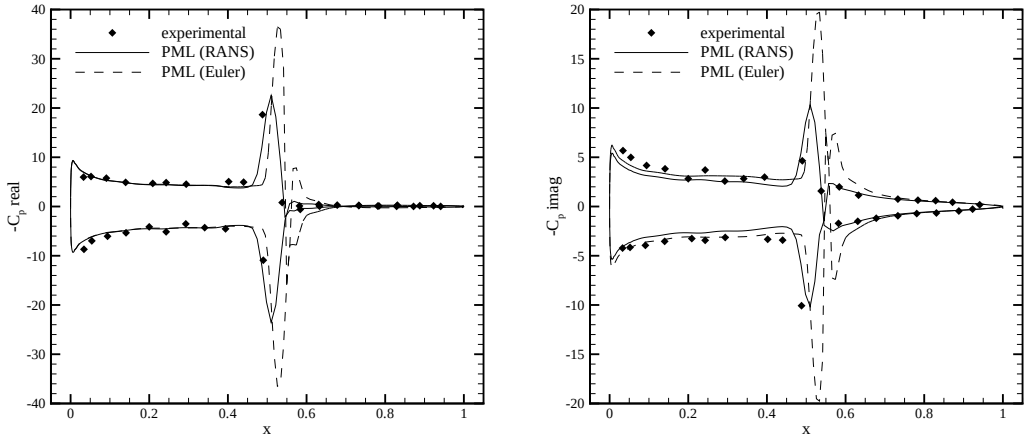
4.1.7 Unsteady turbulent flow - AGARD CT8

The AGARD CT8 case from Ref. [117] is a transonic case for a NACA 64A010 aerofoil, at flow conditions $M_\infty = 0.8$, $\alpha = 0°$ and Re=12.5 million. The forced pitching motion is around the quarter chord, with a reduced frequency of $k = 0.1$, a mean incidence of $\alpha_0 = 0°$ and a pitching amplitude of $\alpha_a = 0.5°$. Five motion cycles with 128 steps each are simulated on a point domain of 29498 points, 340 of which make up the solid wall. This point distribution also has a similar format to that shown in Fig. 4.1(b); and the minimum normal spacing is 6×10^{-6}c. With a CFL number of 60, the linear solver needed, on average, four iterations per pseudo-time step. There were typically 179 pseudo-time steps per real-time step, with a pseudo-step convergence level of 10^{-3}. Each real-time step costs 459 work units on average. The steady state pressure contours for the RANS solution is shown in Fig. 4.10(a); and the surface pressure distributions for the RANS and Euler solutions are shown in Fig. 4.10(b). The surface pressure distribution for both models is reasonably well predicted except in the shock region, where the RANS flow solution shows a closer agreement with the experimental data.

The real and imaginary parts of the first harmonic of the pressure coefficient C_p, normalised with the amplitude of the forcing motion, are shown in Fig. 4.11. The RANS results show good agreement with the experimental data; the inviscid results are also good except at the shock region, where the magnitude of the coefficients is too high, and the shock is slightly further downstream than the location in the experiment due to the lack of viscous effects: this difference can also be seen in Ref. [118]. The real and imaginary lift and moment coefficients are displayed in Table 4.6, with the experimental results from Ref. [117] and computed results from Ref. [119], to further evaluate the accuracy of the solution using PML.

Coefficient	Real C_l	Imag C_l	Real C_m	Imag C_m
PML (Euler)	7.318	-3.542	-0.548	-0.132
PML (RANS)	6.618	-2.816	-0.344	-0.195
Ref. [119]	6.619	-3.021	-0.358	-0.176
Ref. [117]	6.795	-3.403	-0.195	-0.314

Table 4.6: Real and imaginary lift and moment coefficients for AGARD CT8 from PML, compared with experimental data from Ref. [117] and computed data from Ref. [119].



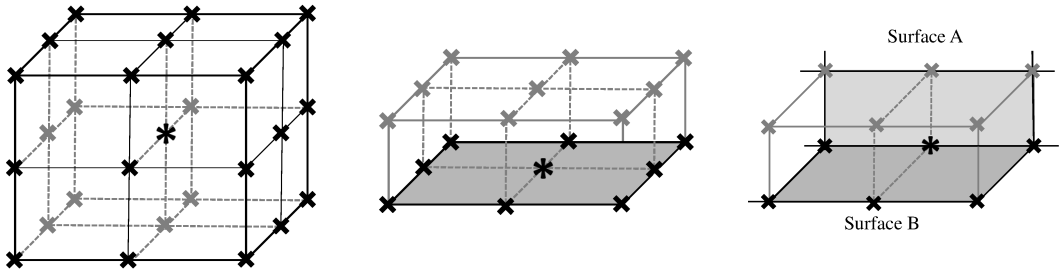
(a) C_p real

(b) C_p imaginary

Figure 4.11: Comparison of real and imaginary parts of surface pressure distribution for AGARD CT8 between PML and the experimental data.

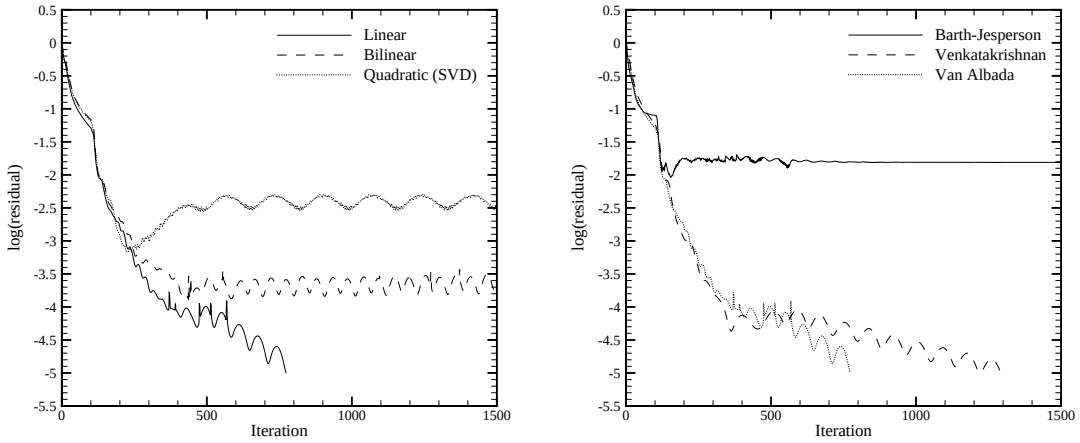
4.2 Three-dimensional cases

The results presented using the three-dimensional meshless solver are inviscid cases over various aircraft wing geometries at subsonic and transonic conditions. The geometries used in the following subsections are the Goland wing, the multidisciplinary optimisation (MDO) wing and the ONERA M6 wing respectively. Each case uses a point distribution obtained from a structured, multiblock grid, from which the PMB results are obtained for comparison. The stencils for PML consist of the 26 points that make up the eight cells that surround the star point, Fig. 4.12(a); boundary stencils generally consist of nine points, including the star point, on the boundary surface and nine interior points, Fig. 4.12(b); while some stencils that form the edge of the geometry, such as along the solid wall and symmetry plane, contain only twelve points, Fig. 4.12(c). Each of the boundary stencils also contains halo points: there is an additional halo for each of the solid wall elements within the stencil, and there is an additional point if the boundary point is to enforce a symmetry or far field condition.



(a) Interior point stencil (b) Boundary point stencil (c) Point stencil between two boundary surfaces

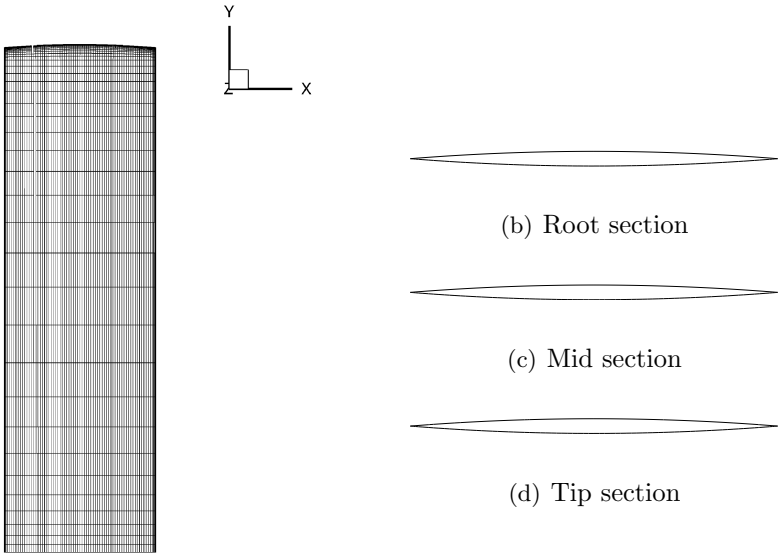
Figure 4.12: Stencils used by PML for three dimensional flow solver validation.



(a) Convergence histories using polynomial shape functions of increasing size (b) Convergence histories using various flux limiters

Figure 4.13: Comparison of convergence histories using shape function reconstructions and flux limiters for the ONERA M6, inviscid test case.

There are issues regarding the size of the polynomials used for meshless approximations in three-dimensions. Recall, from Section 2.2.1, that the polynomials can be of size $m = 4, 7$ or 10 for a linear, bilinear or quadratic reconstruction in three-dimensions respectively. The least squares matrix becomes ill-conditioned at these increasing sizes (and singular for a quadratic reconstruction); and so, instead of solving the normal equations, SVD is used to solve the least squares system. This allows higher order polynomials to be evaluated, but the stencils obtained from mesh connectivities can be too small for an accurate reconstruction to be made. For example, Fig. 4.13(a) shows the convergence history using polynomial shape function reconstructions of increasing order for the ONERA M6 case analysed in Section 4.2.3. The linear reconstruction is the only one for which a converged solution to five orders of magnitude is obtained. Thus, to use the mesh connectivities and be consistent with Section 4.1, the shape functions are evaluated only to linear order in this section.



(a) Goland wing planform with quadrilateral elements

(b) Root section

(c) Mid section

(d) Tip section

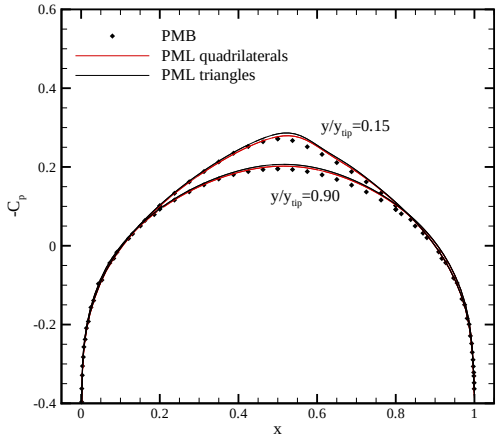
Figure 4.14: Goland wing planform with quadrilateral elements and cross sections.

There are also issues regarding the flux limiters, which are more prevalent in three-dimensions. As Fig. 4.13(b) shows for the ONERA M6 test case, the Barth-Jesperson limiter often fails to give suitably converged results, which is a well known defect of non-differentiable limiters [101]; the Venkatakrishnan limiter is also compared, which converges fully, but is much less efficient than the Van Albada limiter. As a result, the Van Albada limiter is used for all three-dimensional cases in this work.

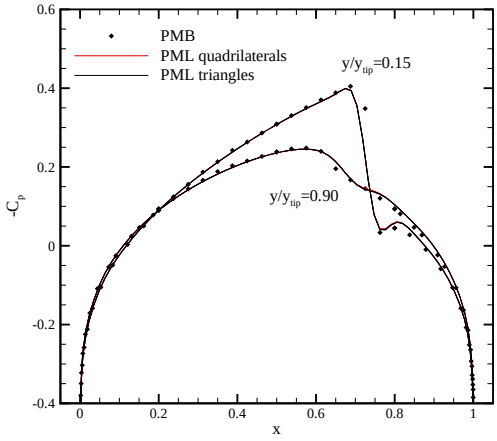
4.2.1 Goland Wing

The Goland wing [120] is a symmetric model wing, having a chord of 1.8266m and a span of 6.096m. It is rectangular and cantilevered, with a constant cross section defined by a 4% thick parabolic-arc aerofoil, Fig. 4.14. The Euler equations are solved at subsonic and transonic flow conditions using two point distributions: one for which the boundary surface is made up of quadrilaterals, and one for which the boundary surface is made up of triangles. Each computational domain is made up of 333705 points, with 6321 on the solid wall, and a first point wall normal spacing of 1×10^{-3}. The quadrilateral surface, as shown in Fig. 4.14(a), contains 6240 elements; while the triangular surface contains exactly double at 12480 elements, as the triangles are formed simply from the diagonals of the quadrilateral elements. The ability of the flow solver to be able to deal with both types of surface element is necessary for Chapter 5, when a mixture of element types is used when boundaries of separate bodies may intersect.

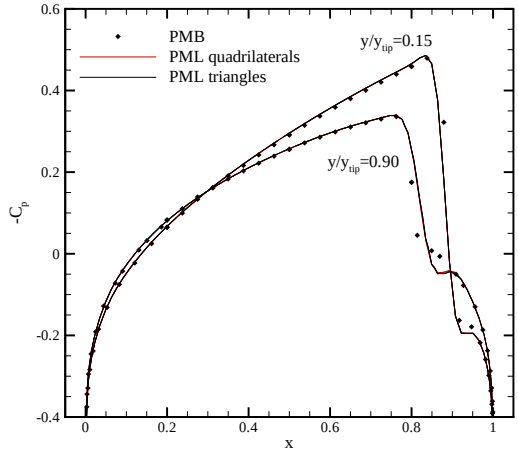
Representative surface pressure distributions at zero degrees angle of attack are shown in Fig. 4.15 at three freestream Mach numbers, and two locations in the spanwise



(a) $M_\infty = 0.875$, $\alpha = 0°$



(b) $M_\infty = 0.9$, $\alpha = 0°$



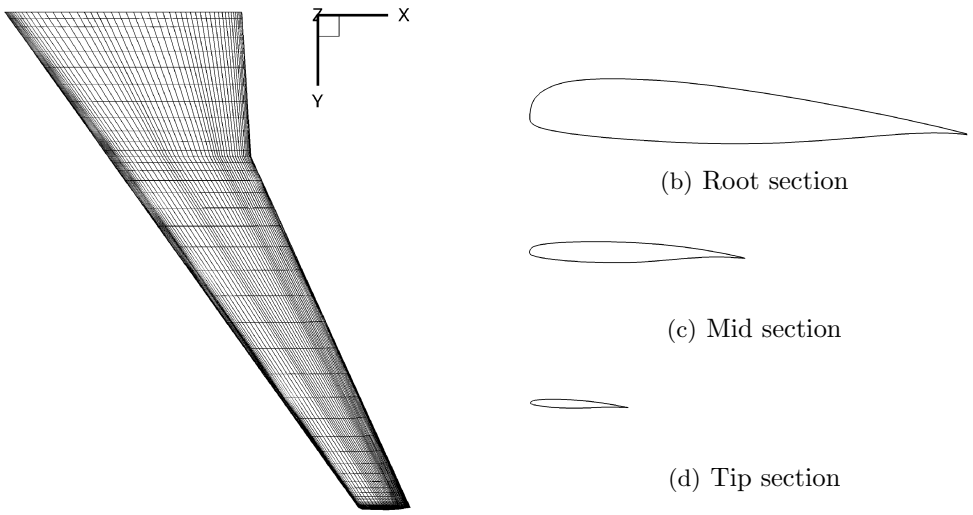
(c) $M_\infty = 0.925$, $\alpha = 0°$

Figure 4.15: Pressure coefficients for the Euler equations using quadrilateral and triangular elements at three freestream Mach numbers and two spanwise locations for the Goland wing.

direction close to the wing root ($y/y_{tip} = 0.15$) and wing tip ($y/y_{tip} = 0.9$). It can be seen from Fig. 4.15(a) that the flow is subsonic at $M_\infty = 0.875$; while higher Mach numbers of $M_\infty = 0.9$ and $M_\infty = 0.925$ cause a transonic flow as seen in Figs. 4.15(b) and 4.15(c) respectively. For both transonic freestream Mach numbers a strong shock wave is formed near the wing root, weakening towards the tip. There is negligible difference between the pressure coefficient plots in Fig. 4.15 for the domains using quadrilateral and triangular elements with PML; and there is good agreement with the finite volume solver PMB.

4.2.2 MDO Wing

The multidisciplinary optimisation (MDO) wing [121, 122] is a highly flexible, commercial transport wing designed to operate in the transonic range. It has a span of 36m and a thick supercritical section. The planform of the wing, including the surface discretisation using quadrilateral elements for the CFD simulations, is shown in



(a) MDO wing planform

(b) Root section

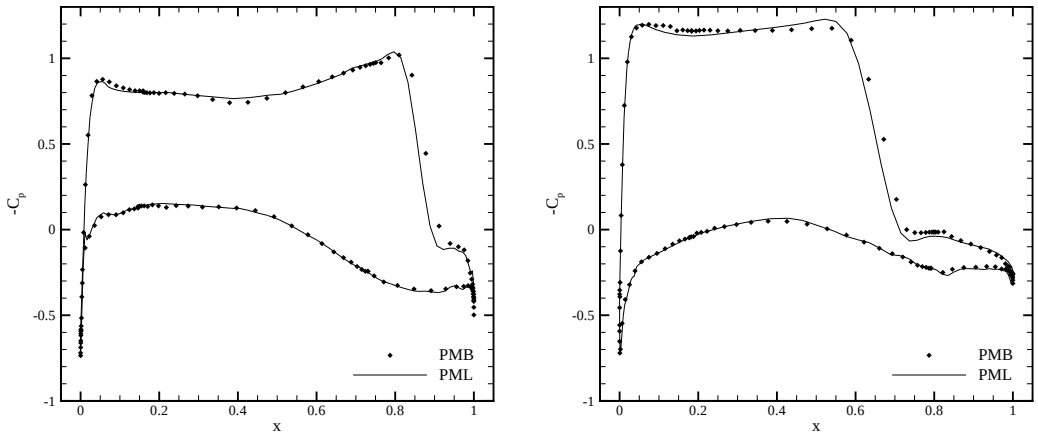
(c) Mid section

(d) Tip section

Figure 4.16: MDO wing planform and cross sections.

Fig. 4.16(a); and the non-symmetric cross sections at the root, mid and tip locations are also given in Fig. 4.16. A computational domain of 283233 points, with 6869 on the solid wall is used for the solution of the Euler equations. Due to the virtual design of the MDO wing, experimental data are not available [123]. The flow is simulated at a fixed transonic freestream Mach number of 0.85 and zero degrees angle of attack.

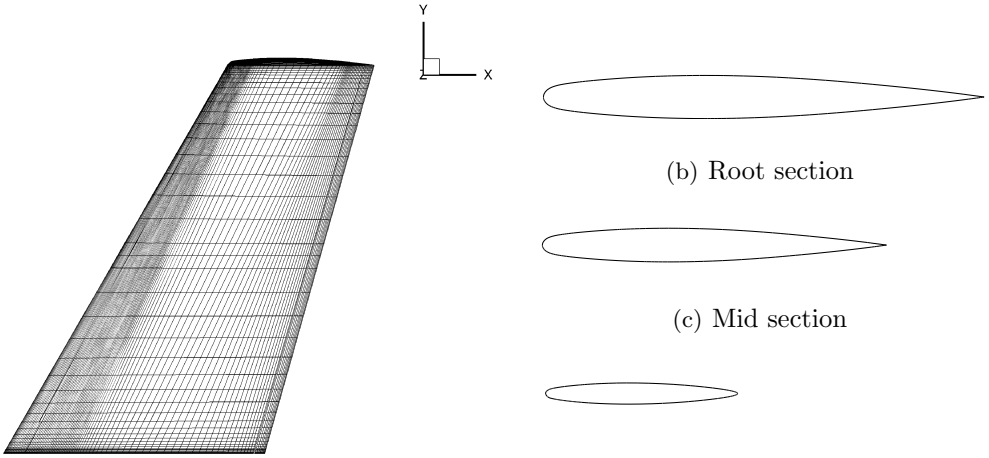
The pressure coefficient at the position $y/y_{tip} = 0.4$ is shown in Fig. 4.17(a), and at $y/y_{tip} = 0.9$ in Fig. 4.17(b). These results are compared with results obtained from PMB and show good agreement.



(a) Pressure coefficient at $y/y_{tip} = 0.4$

(b) Pressure coefficient at $y/y_{tip} = 0.9$

Figure 4.17: Comparison of computed inviscid pressure coefficients with PMB for the MDO wing at M_∞=0.85 and α=0°.



(a) ONERA M6 wing planform

(b) Root section

(c) Mid section

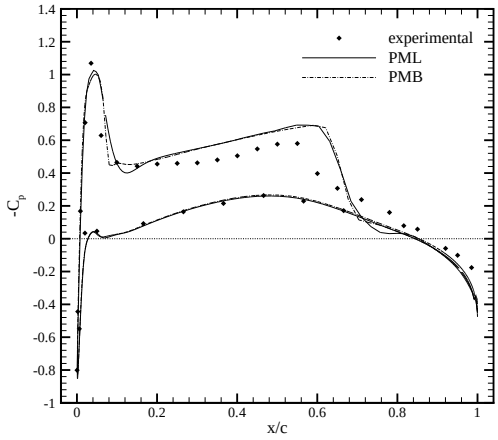
(d) Tip section

Figure 4.18: ONERA M6 wing planform and cross sections.

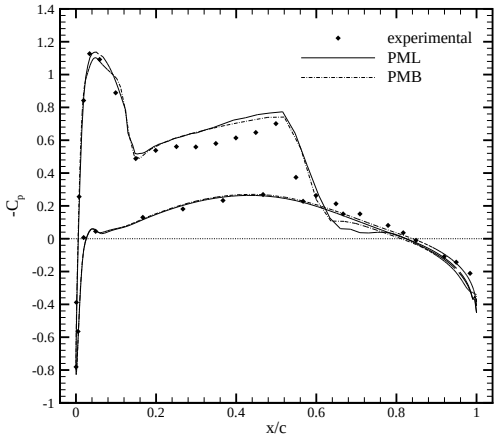
4.2.3 ONERA M6 Wing

For the third test case, we consider the flow over the ONERA M6 wing at transonic conditions. The M6 wing is a semi-span wing, with a symmetric aerofoil section, leading edge sweep angle of 30°, an aspect ratio of 3.8, and a taper ratio of 0.562. The cross sections of the wing at the root, mid and tip sections are given in Fig. 4.18. The flow problem we consider has a freestream Mach number of 0.84 and an incidence of 3.06°. Experimental data for this wing are available in Ref. [124] at test case 2308.

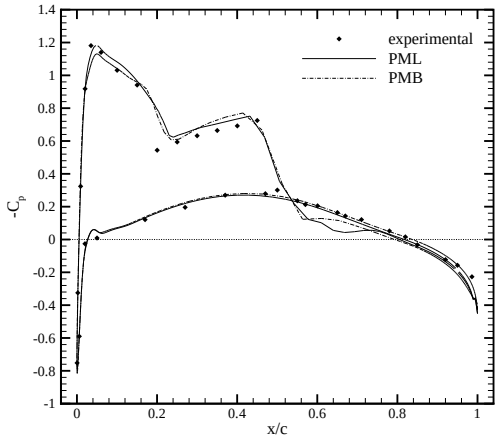
The PML and PMB computations are performed with a point distribution of 1200452 points, of which 10907 are on the solid wall; the planform of the wing can be seen in Fig. 4.18(a). Figure 4.19 shows PML surface pressure coefficient comparisons, with PMB and experimental data, at spanwise locations $\eta = 0.2$, 0.44, 0.65, 0.8, 0.9 and 0.96. A good overall agreement between computed and experimental results can be observed; though, as expected, the PMB and PML results show more agreement as they are inviscid computations. For $\eta = 0.2$, shown in Fig. 4.19(a), the lower surface pressure coefficients agree well with the data, and there are two shock waves along the chord on the upper surface. The forward shock is well predicted; the second shock wave is predicted slightly downstream of the experimental shock location, which is typical of inviscid methods for this case, as the PMB results agree. At $\eta = 0.44$, shown in Fig. 4.19(b), the shock locations have begun to coalesce; and the leading edge suction peak is well predicted. The shocks move even closer together, further along the wing, as is shown in Figs. 4.19(c) and 4.19(d) at $\eta = 0.65$ and 0.8 respectively. At $\eta = 0.9$ and $\eta = 0.96$, shown in Figs. 4.19(e) and 4.19(f), the two shocks have merged to form a single, relatively strong shock wave near 25% of the chord. Here the shock is sharply captured, and the calculated pressures again agree well with PMB and reasonably well with the experimental data.



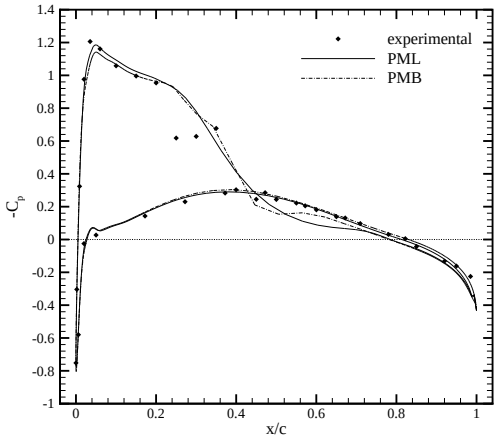
(a) Spanwise location $\eta = 0.20$



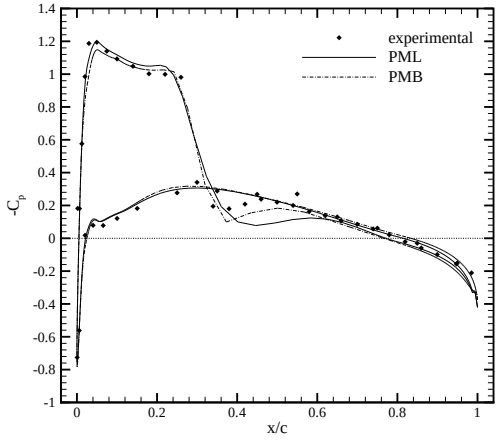
(b) Spanwise location $\eta = 0.44$



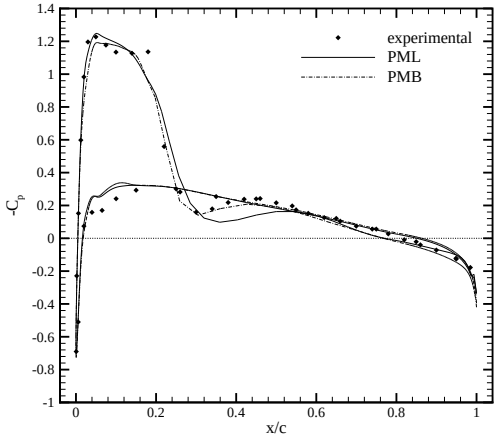
(c) Spanwise location $\eta = 0.65$



(d) Spanwise location $\eta = 0.80$



(e) Spanwise location $\eta = 0.90$



(f) Spanwise location $\eta = 0.96$

Figure 4.19: Comparison of computed inviscid pressure coefficients with PMB and experiment for the ONERA M6 wing at M_∞=0.84 and α=3.06°.

Part II

Development for Multibody Systems

Chapter 5

Meshless Preprocessor

5.1 Scheme requirements

The meshless method, as described in Chapter 2, uses clouds of points to approximate the derivatives in the partial differential equations. Although we are spared many of the difficulties associated with complicated grid generation for finite volume calculations, which is indeed the major benefit of the method, the selection of appropriate stencils, which will give accurate results and are small enough for efficient calculations, is not trivial.

In this chapter a preprocessor method is outlined that calculates the stencils required for the flow solver described in Part I. Section 5.2 outlines the steps involved in the preprocessor method; Section 5.3 deals with the procedure for two-dimensional point distributions, with results presented in Section 5.4; and Section 5.5 describes the method in three-dimensions, with corresponding results presented in Section 5.6.

The focus of this work is on a stencil selection method that is to be used on a given set of points. The points should be arranged so that the geometric features are well resolved, and there are sufficient points to resolve flow phenomena associated with the Navier-Stokes equations such as boundary layers. There are point generation methods available in the literature [125–128] that could be used with the method described; but, instead, we use points obtained from structured grids, such as those in Fig. 4.1(a) due to their availability; though the method could be applied to unstructured grids if necessary. The grids used to solve the Navier-Stokes equations can be highly anisotropic in nature, so are stretched where rapid variation of the flow variables is expected, such as near solid walls or in wake regions. The use of such distributions ensures that the flow in these regions is resolved at relatively low cost. These grids are generated around each body, or component of a body, individually; and then they are allowed to overlap one another to cover the physical domain, so that, in this respect, the approach mimics the Chimera method. The resultant overlap of the point distributions form a single computational domain on which the solver directly operates; so, as there is no

interpolation between the various input point distributions required, the method can be potentially advantageous over the Chimera method since there is no need for difficult interpolation cell identification, and there is no restriction on the relative resolutions of the overlapping input domains. The freedom associated with the method regarding these issues is particularly advantageous for performing moving-body simulations; such problems are accommodated by moving the component grids of each body separately. The stencil selection procedure is designed to work only on the point locations produced by the overlap of the point distributions; whether these are the best point locations for the flow solution is another issue, which is not dealt with in this work.

The overlap of point distributions means that points from one input domain may fall inside the solid boundary of another input domain: such points must be blanked and removed from the final computational domain. In addition, some moving-body problems are characterised by the geometry itself changing during the simulation: control surface deflection is such an example. This means that it is necessary for the stencil selection to be able to deal with intersecting solid wall boundaries if it is to be used in such cases. A procedure that can select stencils from input grids in which the bodies themselves overlap will also make the point generation stage even simpler; so an aircraft geometry can be constructed by the intersection of its component geometry, such as the fuselage and wing.

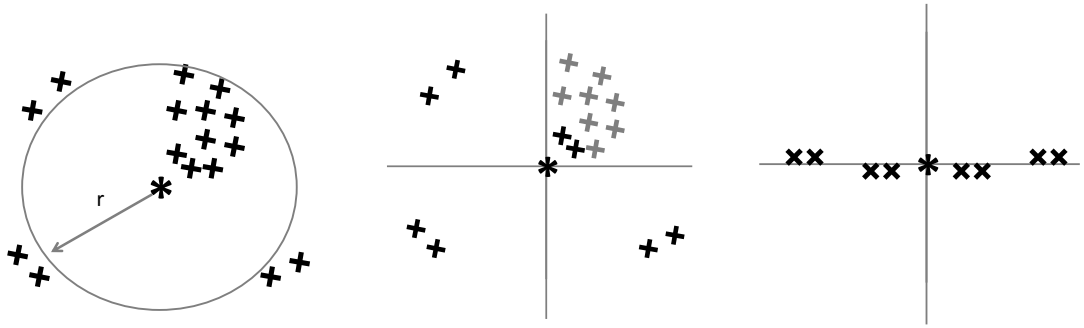
For the method to be useful as an engineering tool, it must satisfy a number of requirements: robustness, automation and speed.

1. It is vital that a computational domain made up of multiple point distributions and bodies has robust stencils for the flow solver. This means that stencils must conform to the geometry of the problem, and be as accurate as possible on the given distribution of points: failure to do so can result in slow convergence, poor solutions and even solver failure.

2. The selection must be automatic: if it requires a large amount of user input then the advantages of a meshless method are lost. This is especially true in the case of a moving body simulation in which multiple stencil selection stages must be made.

3. The repeated use of the stencil selection in moving-body simulations is also the reason why the preprocessor must be fast. A large overhead compared with the flow solver time may mean that alternative methods are preferable.

The meshless points are used to approximate the flow solution as accurately as possible; and the quality of the stencils is ultimately judged by the accuracy with which the flow can be calculated. Unfortunately, the functions to be approximated are unknown before the stencils are to be selected, so only geometric considerations of the point locations are available in the selection. A slightly modified method of selecting



(a) Circular neighbour (b) Quadrant criterion (c) Poor stencil with quadrants

Figure 5.1: Common neighbourhood concepts based on geometric criteria.

stencils for unsteady, moving-body simulations using geometric considerations *and* the flow solution from the previous real-time step is explored later in Chapter 6.

As mentioned in Chapter 1, the strict criteria that must be satisfied by the stencils, without in any way considering the form of the solution that will be obtained, are that:

- there must be enough points chosen for each of the stencils so that the least squares approximation can take place;
- the stencils must be physically large enough for sufficient stencil overlap to occur to prevent discontinuities;
- and the points must not be arranged so as to result in a singular, or very poorly conditioned, least squares matrix.

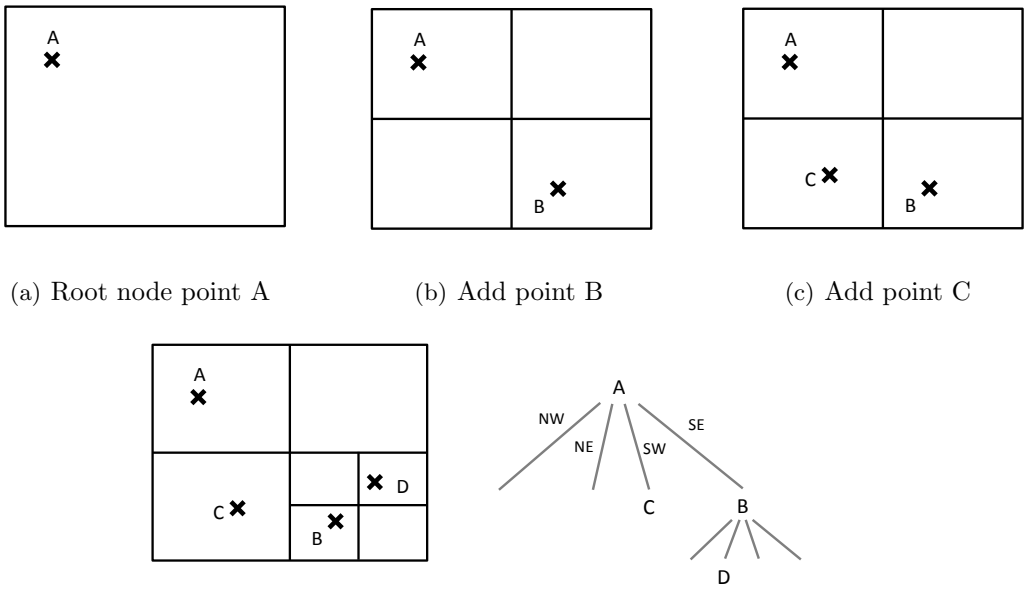
Intuitively, as described in Ref. [46], the stencils should contain a set of neighbouring points, surrounding the star point, with a "centre of gravity" close to the star point location of the stencil. Reference [81] presents some of the most common neighbourhood concepts that are based on geometric criteria; two of these are

- Circular neighbour. To a point $\mathbf{x}_i$, the neighbourhood is defined as all the points $\mathbf{x}_j \in \Omega$ that satisfy

  $$|\mathbf{x}_j - \mathbf{x}_i|_2 < r$$

  The circular neighbourhood, however, does not guarantee that the neighbouring points will be distributed around the star point, as all of the neighbours may lie on one side of the stencil, as in Fig. 5.1(a); and the exact choice of r is also a problem as will be shown below.

- Four quadrant criterion. This is very similar to the circular neighbourhood, but ensures that points are chosen within each geometric quadrant surrounding the star point. Thus, it guarantees that there will be points in all directions relative to a coordinate system, Fig. 5.1(b), making it less likely to give poorly conditioned stencils, though as Fig. 5.1(c) shows, it is not impossible for this to happen.

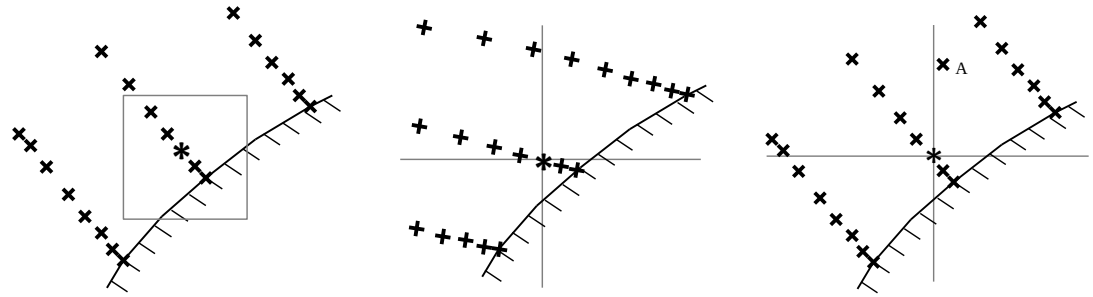


(a) Root node point A (b) Add point B (c) Add point C

(d) Add point D (e) Tree structure

Figure 5.2: The partitioning process of the quadtree.

For the efficient construction of stencils using such concepts, it is preferable for appropriate search algorithms to be employed in the search for potential candidate points. The quadtree [129] is one of the most popular, which is a tree data structure in which each internal node has exactly four children. The first point that is added to the tree is called the root node; and the four pointers associated with this node, that point to the four branches below, are named northwest, northeast, southwest and southeast (in Fig. 5.2 these are labelled NW, located top left; NE, located top right; SW, located bottom left and SE, located bottom right, respectively). The quadtree then partitions the space recursively by dividing it into four subregions. The position of the next node to be added to the tree determines to which of these pointers it is assigned. Figure 5.2 shows a simple example for four points labelled A, B, C and D. Figure. 5.2(a) shows the root node, point A, in the domain; Fig. 5.2(b) shows point B, which lies in the southeast quadrant of the root; point C is added next, which belongs to the southwest quadrant of the root, as is shown in Fig. 5.2(c); while Fig. 5.2(d) shows that point D belongs to the northeast quadrant of point B. The tree structure for this partitioning is shown in Fig. 5.2(e). A search region is defined, which is usually in the form of a square or rectangle, and the points that are located within this region form the list of candidate points. The partitioning method means that searching for points within the prescribed search region does not require checking *every* point in the domain: only the branches of the quadrants that overlap the search region. The search typically operates in $Nlog(N)$ time, where N is the number of points, as opposed to N^2 time required by a brute force search.



(a) Poor choice of search region (b) Poor quadrant selection near boundary (c) Point A would be part of the stencil of the star

Figure 5.3: Candidate point search and stencil selections using nearest neighbour algorithms and anisotropic point distributions.

The first attempt at creating a meshless preprocessor was in two-dimensions, and used no initial connectivity data. Quadtree searches were used to find the candidate stencil points, and a quadrant procedure was used to select the stencils. The search region used for the quadtree was determined using some initial reference length to form a box around the point for which the stencil was to be constructed. If too few candidate points were found in the original method, then the search region was increased and the search starts again; if too many points were found, then the search region was decreased. This process is straightforward if the point distributions are well-spaced or isotropic throughout the domain. If this is not the case, then the anisotropy of the points means that there is no clear way to determine if the number of candidate points chosen is sufficient for an accurate stencil. For example, Fig. 5.3(a) shows a poorly chosen search region for an anisotropic point distribution, as the points that lie within are collinear, and so would cause a singular least squares matrix. This region can be increased, though it cannot be determined exactly when the number of points obtained is sufficient.

As well as the lack of robustness for the first requirement that the method must have to be a practical engineering tool, this situation is also not ideal for the third requirement. Not only are repeated searches and stencil checks expensive, but the quadtree procedure gets less efficient as the search regions are increased. If a square search region is used, then there may be hundreds of candidate points found to ensure a well-conditioned least squares matrix. This would be costly for the quadrant procedure, and, as only a small fraction of points are needed in the stencil, much of the effort spent in finding and checking these points is unnecessary. If this is global, then it is unnecessary work in regions of the domain where the point distributions are isotropic. Non-square search regions could be used in parts of the domain where the points are arranged in an anisotropic manner, while square regions could be reserved for the isotropic point regions; however, this would require additional user input, and hence, the procedure would not be fully automatic as the second requirement states.

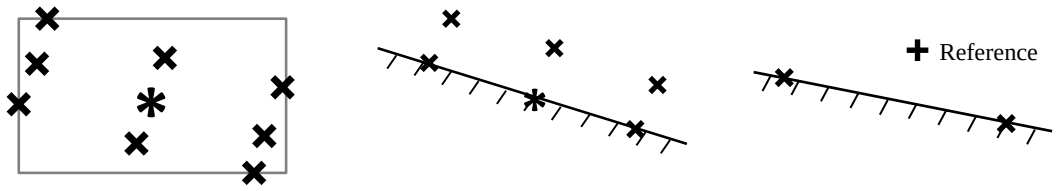
In addition to these problems, the quadrant procedure itself is not ideal for use with overlapping anisotropic point distributions, as even if a satisfactory number of points is found, the resulting stencils are not guaranteed to be adequate. The isotropic nature of the quadrant procedure means that it has problems for cases, such as that in Fig. 5.3(b), in which the nearest points in the top-right quadrant for a star point in the boundary layer lie far from the boundary layer region. The method could be restricted so that only point distributions are used in which the points lie exactly perpendicular to the boundary wall; however, we wish to avoid the need to impose conditions on the input domains, as this will require more effort in the point generation scheme. It is clear that the points chosen is dependent on the fixed coordinate system, which is not ideal. Accuracy is further compromised if a point from one grid falls into a region of anisotropy of another as in Fig. 5.3(c). The star point would select point A in its stencil as it is the closest in its quadrant; this would again mean that a point far outside the boundary layer would be influencing the solution at the star.

The anisotropy of the point distributions, which are necessary to solve the Navier-Stokes equations, means that a stencil selection method must be developed with this property taken into account; and, in summary, the two major problems with stencil selection methods on point distributions of this kind are in:

- the search for good candidate points;
- the selection of points from the candidates that reflect the anisotropy.

As a result of these issues, the notion of a "pure" meshless scheme was abandoned for the preprocessor stage; instead, the original grid connectivities can be used, and as such a semi-meshless method is obtained. The difficulties with finding an ideal search region are avoided if the limits of the original grid stencils are used as an initial search region, as in Fig. 5.4(a). They also help to make the point blanking process much more robust and efficient, as will be described in Section 5.3.2. This now means that all of the requirements above are satisfied for the search: there is no further user input, computational time is reduced because only one search is necessary, and robustness is guaranteed because the grid stencil itself could be used if no additional points are found. There are still questions regarding the final stencil selection from the candidate points, but using the initial connectivity as a guide can also help in this process, especially regarding the anisotropy of the local points; the issues relevant to this stage are addressed in Section 5.3.3. In using a semi-meshless approach an additional requirement, to the previous three, is placed on the preprocessor scheme.

4. Despite using the grid connectivity as a guide, we still want to retain the simplicity of the meshless method globally. This means that we wish to avoid operations often used with finite volume methods, such as cut-cell methods, hole cutting, or putting precedence on a single grid, or regions of several grids, when constructing the stencils.



(a) Interior point connectivity and stencil bounding boxes

(b) Connectivity for boundary point stencil

(c) Boundary edge reference point inside domain

Figure 5.4: Required input for preprocessor.

5.2 Preprocessor steps

There are four primary operations that the developed method performs to construct the stencils required by the meshless flow solver.

1. Check if the solid wall boundaries from separate component grids overlap in any way. This can be the case if we wish to construct a geometry from individual component grids. If overlap occurs then the boundaries must be altered accordingly.

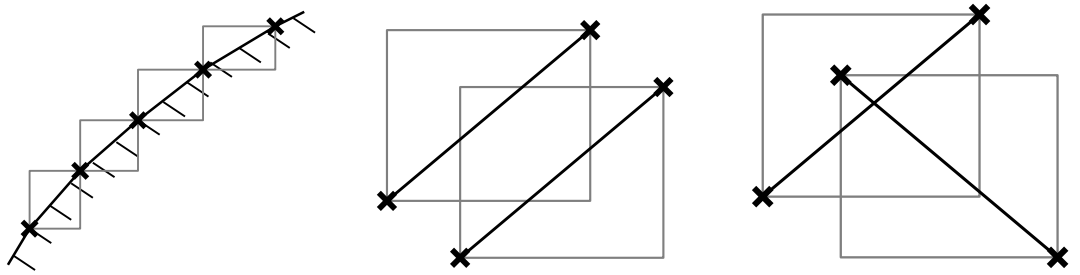
2. Identify the points that lie within a solid body, which can occur when the grids overlap. These hidden points are blanked and removed from the flow computation.

3. Select the meshless stencils for the remaining points.

4. Make sure that the resultant stencils respect the boundaries. This means check that no stencils are formed that contain points that lie on the opposite side of a solid wall.

The boundary definition, required in steps 1, 2 and 4, is an important stage of the process. The bodies that make up the computational domain can be complex, and it is essential that the selected stencils conform to the geometry of the problem. This is done with a combination of flags that classify the points at various stages of the preprocessor, and is based around tree algorithms for efficiency. The stencil selection of step 3, must work with anisotropic point distributions, and satisfy the four requirements outlined in Section 5.1. Each of the operations above is described for two and three-dimensional geometries in the following sections.

The full user input is as follows: first, a list of points with coordinates; second, the stencils that form the original grid connectivity (for both interior and boundary points as shown in Fig. 5.4(a) and Fig. 5.4(b), respectively, for a case in two-dimensions); third, a list of boundary elements that form the closed geometry, with the boundary type, and a reference point, for each element, that lies on the inside of the domain in that grid; this reference point is used to establish the inner normal vector for when the vector perpendicular to the line segment is found, Fig. 5.4(c).



(a) Edge bounding boxes

(b) Bounding boxes intersect but edges do not

(c) Bounding boxes intersect and edges intersect

Figure 5.5: Tests to determine if edges intersect using bounding boxes.

5.3 Preprocessor in two-dimensions

5.3.1 Redefining boundaries

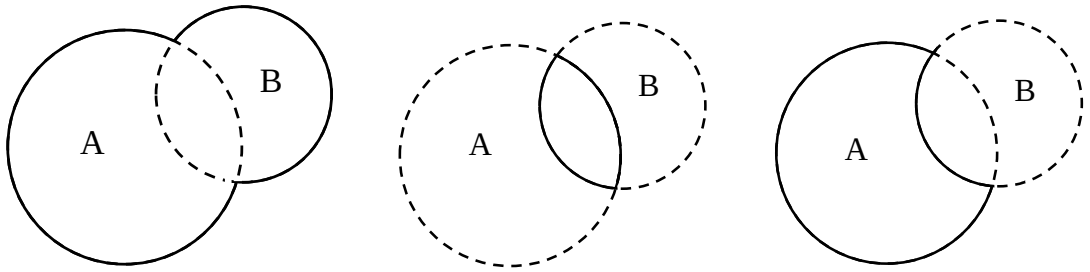
The first step is to identify and fix any boundary overlap. To do this the boundary edge of one component grid is checked with each boundary edge from the other grids, using a geometric intersection algorithm, to see if they intersect. As it is expensive to check every edge against every other edge in the domain for an intersection directly, we use the method described in Ref. [130] to reduce the number of edges that must be checked. Bounding boxes are formed around each boundary edge so that the edge coordinate limits form the corners of the box, Fig. 5.5(a); and each bounding box from one grid is compared with those from the other grids to see if the boxes intersect. This is done by transforming an object in $\mathbb{R}^D$ to a point in $\mathbb{R}^{2\times D}$, with point coordinates as the limits of the object. A box in two-dimensions would therefore be seen as a point in four-dimensions with coordinates

$$\mathbf{x} = \{x_{min}, y_{min}, x_{max}, y_{max}\}$$

We can then construct a search tree, similar to the quadtree, containing each of these higher-dimensional points. This method is also used in Refs. [131, 132], which are concerned with Cartesian mesh techniques. If all of the boxes lie within a domain Ω, with coordinate limits $\mathbf{x}_{\Omega,min}$ and $\mathbf{x}_{\Omega,max}$, then two box elements a and b will intersect if they satisfy

$$\begin{pmatrix} x_{\Omega,min} \\ y_{\Omega,min} \\ x_{a,min} \\ y_{a,min} \end{pmatrix} \leq \begin{pmatrix} x_{b,min} \\ y_{b,min} \\ x_{b,max} \\ y_{b,max} \end{pmatrix} \leq \begin{pmatrix} x_{a,max} \\ y_{a,max} \\ x_{\Omega,max} \\ y_{\Omega,max} \end{pmatrix} \tag{5.1}$$

The tree is then traversed with the bounding boxes from grid 1 to see if any of the bounding boxes from grid 2 overlap. If this is so, then only these edges must be checked for intersection, which is done using a simple ray intersection algorithm [133],



(a) Cylinder A flag=1, Cylinder B flag=1

(b) Cylinder A flag=0, Cylinder B flag=0

(c) Cylinder A flag=1, Cylinder B flag=0

Figure 5.6: Boolean addition and subtraction of two overlapping cylinders.

to distinguish between the cases illustrated in Figs. 5.5(b) and 5.5(c).

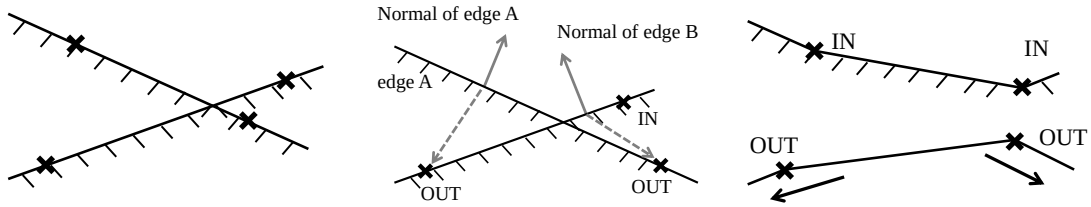
If edges intersect then it is necessary to redefine the boundaries to accommodate the overlap. The new boundaries can be made with either a Boolean addition or a Boolean subtraction. Each body is given a flag to indicate this: 1 for addition and 0 for subtraction. It is possible for a Boolean addition to occur for one body and a subtraction for another. Cases for two circular cylinders are shown in Fig. 5.6.

The overlapping boundaries could be redefined with the addition of either a new point or a new edge. If we were to redefine the boundary by adding a new point, it would be necessary to search for a stencil for this point. This can be very difficult as the number of candidate points can be very small in some regions where boundary overlap occurs; the candidate points found can often lead to very poor stencils, and so there is a greater risk of the solver failing at such points. This problem is avoided completely if, instead, a new element is added where the intersection occurs. To decide which of the boundary points in Fig. 5.7(a) should form the redefined boundary wall a normal test is used, such that

$$\mathbf{n}_{edge}.(\mathbf{x}_{point} - \mathbf{x}_{edge}) \geq 0 \tag{5.2}$$

where $\mathbf{n}_{edge}$ is the normal vector of the boundary edge, $\mathbf{x}_{edge}$ is a point on this edge and $\mathbf{x}_{point}$ are the coordinates of the point on the other element to be tested against. If Eq. (5.2) is satisfied, then the point is located in front of the boundary edge, within the domain; if not, then the point lies behind the edge, possibly inside the wall. Surface normal tests may fail if the body does not have the same normal in the vicinity of the point to be tested: so it will often fail to identify a point located behind the wall if the body is concave, or in front of the wall if the body is convex. In this case, however, the test is applied only to straight line segments that intersect, therefore they are in the same vicinity, so the test is acceptable.

The test is applied using the normal of edge A and both of the points forming edge B. If the result is greater or equal to zero, then the point belonging to edge B is flagged as IN; if it is less than zero then the point is flagged as OUT. The same operation is performed for the normal of edge B and the points of edge A. If a point that belongs



(a) Intersecting boundary edges (b) Identifying IN/OUT points (c) Flagging new interior edges

Figure 5.7: Boundary intersection and reallocation diagrams for two bodies requiring Boolean addition.

to the boundary edge of a body that is flagged for a Boolean addition is flagged as IN, then it is a continuation of the redefined boundary; otherwise, if it is flagged as OUT then it is no longer a boundary point and is flagged as an interior point [1]. The opposite occurs for a body flagged for a Boolean subtraction: if a point on the boundary is flagged as IN, it is flagged as an interior point, else it remains a boundary point.

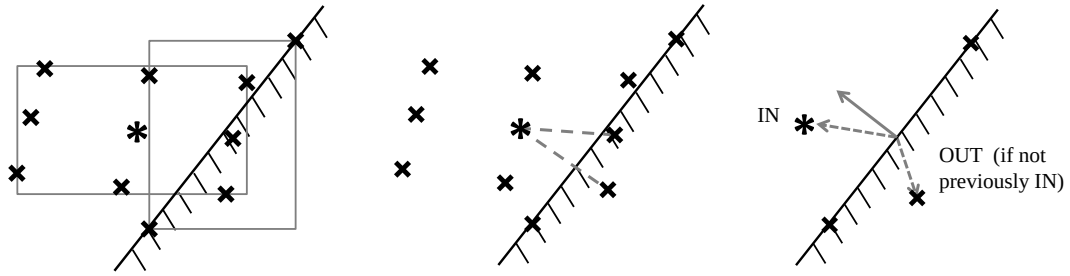
Now that we know which of the boundary points form the redefined wall, it is relatively simple to add a new element between the boundary points that make up the intersection. It makes no physical sense for the new interior points to still have boundary edges; hence, these edges (and those of their neighbours) are removed from the boundary element list. It is also necessary to flag the points of the removed boundary elements as interior, which is done by the use of a simple operation in which the point sharing the same edge element is flagged until there are no boundary points left, Fig. 5.7(c). In this way the boundaries have now been fully altered.

If there are more than two grids that form the final domain then the grids are compared sequentially: so grids 1 and 2 are compared, then the resultant domain is compared with grid 3, then the resultant domain is compared with grid 4, and so on.

5.3.2 Blanking points

The task of identifying the interior points that need to be blanked due to the overlap of the input domains is done next; and, to keep the method as efficient as possible, a similar higher-dimensional search algorithm to that used to identify intersecting boundary edges is employed to identify the interior points that may need to be blanked. This time search regions are defined, using boxes that are formed around the new boundary edges of one grid, and a search is performed with a tree that is made up of the boxes formed around the grid stencils in every other grid, as in Fig. 5.4(a). If a stencil overlaps with a boundary edge, so it satisfies Eq. (5.1), as in Fig. 5.8(a), then some of the points could lie inside a solid body. The normal test of Eq. (5.2) cannot *solely* be used now to test if a point lies behind a wall because of its failure with the cases outlined previously;

[1] It could be that these points now lie inside a boundary wall, they are still flagged as interior for now though, and will be blanked in the next step, outlined in Section 5.3.2.



(a) Box intersection (b) Remove points from stencil (c) Normal test

Figure 5.8: Blanking point operations.

instead, the same ray intersections that are used to identify edge intersections are used, which are much more robust. Thus, for each stencil box that intersects the boundary edge box, rays between the star and every neighbouring point are formed within the original stencil. If a ray intersects with the boundary edge then this neighbouring point is removed from the stencil, as it must lie behind a boundary, as in Fig. 5.8(b). A normal test is then performed to identify whether it is the star or the neighbouring point that lies behind the wall. If Eq. (5.2) is satisfied then it is flagged as IN, if not then it is flagged as OUT *provided* it has not already been flagged as IN, Fig 5.8(c). This system of flags is needed to prevent the blanking of points that lie behind a boundary wall, but are still inside the domain, which can occur, as discussed for a convex body, in Fig. 5.9, in which normal vector B would flag the star point as OUT if it were not for normal vector A flagging the point as IN. As this is a global flag, then if the flag is at any stage set to IN then it cannot be a blanked point. When each stencil is checked in the domain, at the end of this step, all of those points that are flagged as OUT lie inside a boundary wall and, therefore, are outside the computational domain.

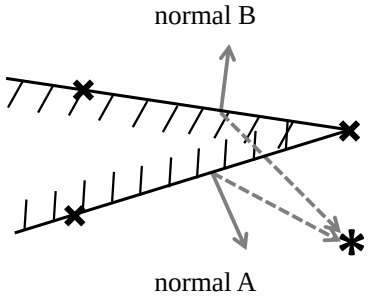


Figure 5.9: Situation with possible multiple flaggings.

A flood operation is then performed, in which the neighbouring points of a blanked point are similarly blanked. This is to ensure that the points that lie inside the boundary, but do not lie near the wall, are identified as such. Neighbours are defined as those points that are within the same stencil: so a "leak" will not occur in the computational domain due to the previous step in which points located on the opposite side of a wall to the star were removed from its stencil. This process is repeated, so neighbours of neighbours are blanked, until there are no points remaining. The blanked points are removed from the flow solver computation by setting the residual for each point to zero, and setting its diagonal block in the Jacobian matrix to the identity matrix.

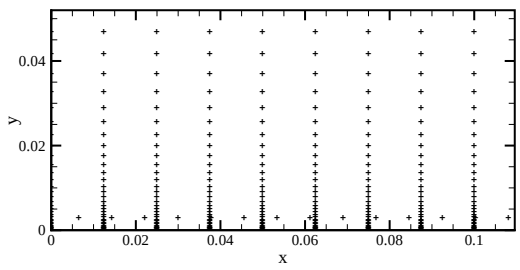
5.3.3 Stencil selection

The previous steps of the preprocessor ensure that the boundaries are defined properly, points within solid bodies are blanked, and that stencils do not intersect any of the boundaries. The interior stencils, however, are still grid dependent, so there is no connectivity between the points from different input grids. The next step is to form stencils, irrespective of the input domains, that effectively link the grids together to form the final meshless domain connectivity for the flow solver.

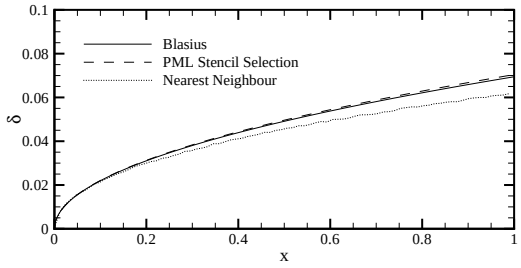
The overlapping of the grids gives us a fixed set of points on which to construct stencils that make the best approximation of the derivatives at the star point of an unknown function Eq. (2.7), approximating the flow variables. As outlined in Section 5.1, it is the anisotropy of the overlapping point distributions that causes the greatest difficulty for the stencil selection. If the function to be approximated has a high gradient in a particular direction, such as in the boundary layer, then it is much more difficult to select accurate stencils from the fixed anisotropic point distributions. An example of this can be seen in Fig. 5.10 for a laminar Navier-Stokes calculation on a flat plate, with a line of points added to the original structured grid in the boundary layer region Fig. 5.10(a). The boundary layer thickness δ is plotted in Fig. 5.10(b) for cases when the stencils are selected using the nearest neighbour/quadrant procedure and the stencil selection method to be described below. The results are compared with the Blasius solution; and it can be seen that the quadrant method, in general, gives lower values of the boundary layer thickness. In addition, despite the influence of weights used in the meshless solver, the curve plotted is not completely smooth. The reason for this can be seen in Fig. 5.10(c) which shows two stencils that are selected using this method: one of the stencils shown is located above the added line of points, and the other is located below. The range of the points within the stencils is spread across the boundary layer region causing the inaccuracies (please note the scale of this plot). In contrast, the stencils selected using the method to be described in this section are more refined in the boundary layer region, Fig. 5.10(d), to capture the high gradient flow, leading to greater accuracy as seen in Fig. 5.10(b).

For stencils to be constructed for accurate flow simulations the points must be chosen so that the worst case (highest gradient) functions that determine the flow are best resolved. To do this, the underlying information about the anisotropy of the grids before the overlap occurs is used in the selection, since these determine what functions can be resolved; this is a different issue from what point distribution is needed to resolve the actual solution function. We quantify the anisotropy by defining the resolving vector $\mathbf{v}$ of a grid stencil, which points in the direction in which the original stencil is the finest, that is, the direction for which a function is best resolved.

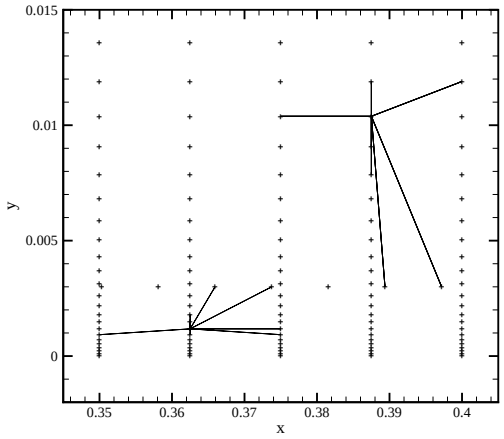
To determine the resolving vector, we use the point locations of the original grid stencil. The resolving vectors are computed, for each point, before the overlap of the



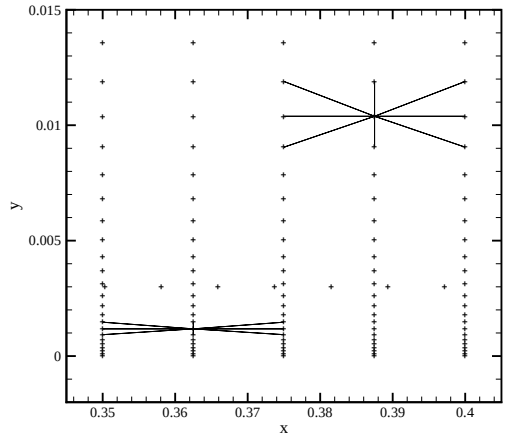
(a) Close up of point distribution for flat plate



(b) Boundary layer thickness using stencil selection methods



(c) Close view of a boundary layer stencil using nearest neighbour (quadrant) method



(d) Close view of a boundary layer stencil using described stencil selection method

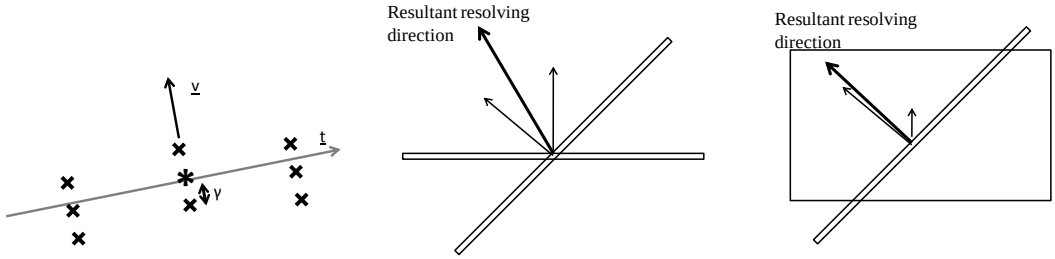
Figure 5.10: Comparison of results using classical nearest neighbour algorithm and presented stencil selection method for solution of the Blasius boundary layer on anisotropic point distributions.

input domains occurs, so the entire initial grid stencil is used in determining it, even if some of the points within the initial stencil turn out to be blanked. This preserves the quality of the resolving vectors for points that lie near a boundary wall of another input domain; it also means that there is no need to recompute vectors during a moving-body simulation, when some points will be blanked or unblanked during the computation. The principal axes of the point cluster of the initial stencil are computed, which are a set of orthogonal vectors, often used in the mechanics of rigid body motion; this requires calculating the moments of inertia of the points, assuming equal mass

$$m_x = \sum_j (x_j - x_i)^2 \qquad m_y = \sum_j (y_j - y_i)^2 \qquad m_{xy} = \sum_j (x_j - x_i)(y_j - y_i) \tag{5.3}$$

where the sum j is over the points in the original grid stencil of point i. The angle of the vector with respect to the x axis is then obtained from

$$\theta = \frac{1}{2} \arctan\left(\frac{2m_{xy}}{m_x - m_y}\right)$$



(a) Resolving vector for a grid stencil (b) Two anisotropic stencils overlap (c) An anisotropic and a more regular stencil overlap

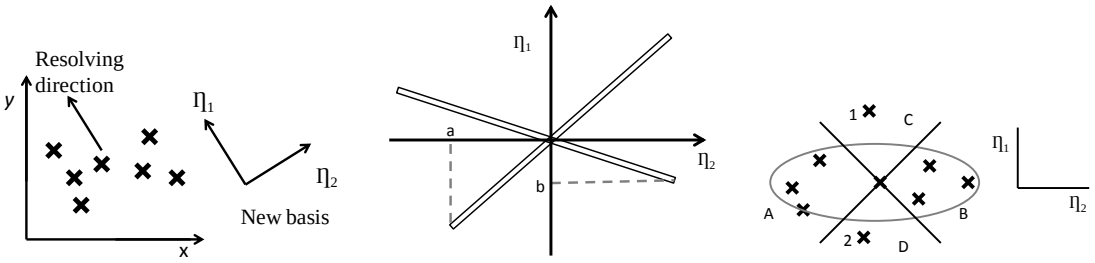
Figure 5.11: Defining the initial resolving vector and method of summation of overlapping stencils.

As the range of arctan is $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$ in radians, then only one of the principal axes is found. The vector found may pass through the points as a line of best fit, as $\mathbf{t}$ in Fig. 5.11(a); if this is the case then the vector perpendicular is the resolving vector $\mathbf{v}$; if not, then the principal axis found is the resolving vector. We can decide on the choice of vector by rotating the points so that the principal axis found points in the y direction. A projection of the points is then made on the y axis, and the largest value, denoted $|d_y|$, is stored. A projection is then made on the x axis, and the largest value, denoted $|d_x|$, is stored. If $|d_y| > |d_x|$ then the resolving vector points orthogonal to the principal axis found, else the resolving vector is the principal axis.

If we define the quantity γ as the distance from the star to the closest point in the original stencil, Fig. 5.11(a), then we can set the length of the resolving vector to vary inversely with γ (or some power p of γ) to reflect the refinement of the point distribution in this direction. Thus,

$$\mathbf{v} = \frac{\hat{\mathbf{v}}}{\gamma^p} \tag{5.4}$$

where $\hat{\mathbf{v}}$ is the unit vector. This means that a small distance γ, as for a boundary layer grid stencil point, will give a large resolving vector; and a point for which the stencil is less refined will give a small resolving vector. When the grids overlap, we need to automatically construct meshless stencils that will reflect the anisotropic nature of *a*ll of the grids to capture the flow accurately. This is done, for each point, by searching amongst the other grids for the stencils that overlap the original stencil of the point in question. The same bounding boxes as in Fig. 5.4(a) are used as the search region, with the quadtree search algorithm outlined in Section 5.1. The points that make up all of the overlapping stencils found, form the list of candidate points from which the meshless stencils are constructed. From each of these overlapping stencils, the one with the lowest value of γ for each grid (so the finest stencil) is identified. It is these stencils that will give the best estimation of the direction of refinement in this region. The resolving vectors of these points, and the point for which the stencil



(a) Defining the new coordinate system

(b) Choosing a and b using stencil projections

(c) Quadrants around resolving direction

Figure 5.12: Defining coordinate system and parameters for the merit function.

is being constructed, are then summed to give a resultant resolving direction for the composite points in that region of the domain. Some examples of the summation of the resolving vectors, with representations of the overlapping original stencils, are shown in Figs. 5.11(b) and 5.11(c). For Fig. 5.11(b) the resultant resolving vector is an average of both of the resolving vectors due to the stretched nature of both of the overlapping stencils; while for Fig. 5.11(c) the resultant direction is more in the direction of the stretched stencil as the other stencil is more isotropic. The higher the value of p in Eq. (5.4), the more influence the finer stencils have on the resultant direction. In this work the value $p = 4$ is used.

In the resultant resolving direction we define a new coordinate system as shown in Fig. 5.12(a). The basis $\boldsymbol{\eta}$ is chosen so that $\boldsymbol{\eta}_1$ lies collinearly to the resultant resolving direction and $\boldsymbol{\eta}_2$ lies orthogonal. Setting the star to be at the origin, we define ξ_2 and ξ_1 to be the coefficients of $\boldsymbol{\eta}_2$ and $\boldsymbol{\eta}_1$ respectively, that form the coordinates of each candidate point in this basis. Then a merit function ψ is defined, which rates each candidate point in terms of the required direction and refinement, by balancing the orthogonality and the distance of the points from the star,

$$\psi = \frac{\xi_2^2}{a^2} + \frac{\xi_1^2}{b^2}$$

This equation is similar to the equation for an ellipse, with the constants, a and b, as the semi-major axis and semi-minor axis. For each of the stencils in the overlap, the largest projection of the vector from the star to each neighbouring point in the original connectivity onto η_1 is found. From each of these projections the smallest is used as the value for b. Similarly, the largest projections onto η_2 are found for each overlapping stencil, and the smallest of these is used as the value for a, Fig. 5.12(b). It was found that this was the best way to keep the stencils fine, but well-conditioned. In this system the conditioning of the least squares matrix is further improved, as in Section 5.1, by forming four quadrants around the axis system as shown in Fig. 5.12(c) from which points are chosen independently, so the merit function is not compared between points lying in different quadrants.

For the meshless scheme in two-dimensions with a quadratic reconstruction, we need at least seven points in each stencil for the system to be overdetermined, though more are preferred to improve stability. Using Fig. 5.12(c), this is done by selecting one point from quadrants A and B, each with the smallest values of ψ from their respective lists. Then, from each quadrant A and B, we select another two unique points: one with $\xi_1 > 0$, and one with $\xi_1 < 0$. Thus, we have six points from these quadrants total, which are relatively spread. The maximum value of ψ from these six points is denoted ψ_{max}. To retain accuracy we then take one point from each of the quadrants C and D, each with the lowest value of ψ, provided these values are less than ψ_{max}. This condition is to preserve the refinement of the meshless stencils; otherwise, it is possible that the nearest points within C and D are very far away, which would destroy the anisotropy. This can be seen in Fig. 5.12(c); the ellipse shown is at ψ_{max}, and since the value of ψ for the points in C and D exceed this, they are not selected, and the stencil would remain at six points, or seven including the star.

The scheme can thus be defined as a quadrant scheme with the resolving vector defining the coordinate system, and a merit function used to provide the anistotropy required to solve the Navier-Stokes equations. This merit function is beneficial in that the strength of the level of directional refinement of the final meshless stencils is dependent on the original connectivities. For highly stretched stencils near the boundary $a \gg b$, and so the orthogonality of the points will have more of an influence on ψ: this will lead to anisotropic meshless stencils. On the other hand, if the overlapping stencils are more isotropic then $a \approx b$ and there is less refinement. In fact, for the special case $a = b$, ψ resembles the equation for a circle and there is no refinement at all. It is for this case that the method effectively reduces to the nearest neighbour algorithm with quadrants.

5.3.4 Final boundary check

Once the stencils are created, the final step is to check that all of the stencils formed respect the boundaries. This step is necessary because the previous boundary check in Step 2 of Section 5.2, used stencils from the original grid connectivity and was to identify blanked points; while the final stencils are formed using points from all of the input domains. It is possible for the points of a stencil to lie on the opposite side of a boundary wall where there are corners or sharp edges, such as the trailing edge of an aerofoil. If a point within a stencil lies behind a solid wall to the star, then it is simply removed from the stencil. This check is done using the same method as in Section 5.3.2, though now each boundary edge must be checked with all of the stencils in the domain, not just those from other grids. Consequently, the boundary check in this step is slightly more expensive than previously in Step 2, but there is now no point blanking scheme.

5.4 Results in two-dimensions

Six test cases that demonstrate the preprocessor in two-dimensions are presented; each case consists of two input geometries, which may not reflect a real life aerospace application, but are designed to demonstrate the scheme. The first and second cases use a biplane configuration to solve the Euler equations; the third, fourth and fifth cases are also biplane configurations, for the solution of the laminar equations for the third, and the RANS equations for the fourth and fifth cases; the sixth is a prototype control surface deflection case, which demonstrates the ability to solve the RANS equations with fully intersecting geometries.

5.4.1 Subsonic steady state flow

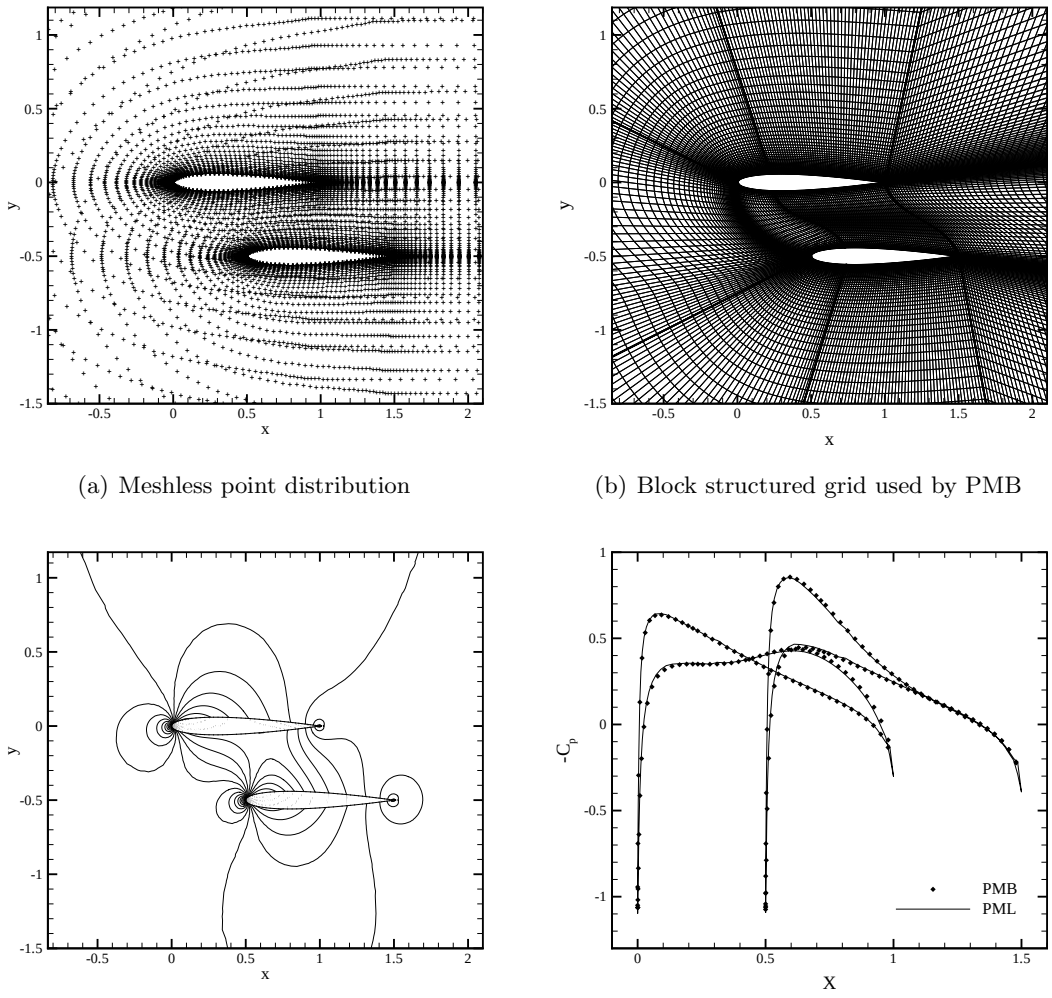
The first biplane configuration is formed by the overlap of two NACA 0012 aerofoils of 9408 points, each with 96 on the solid wall boundary. The leading edge of the upper aerofoil is located at the origin; while the leading edge of the lower aerofoil is at (0.5,-0.5). The resultant fixed point distribution obtained from the overlapping grids is presented in Fig. 5.13(a). The blanked points are not shown, and so it is on these points that the stencil selection is made. The inviscid, steady state pressure contours at a freestream Mach number of 0.5 and zero degrees angle of attack are presented in Fig. 5.13(c).

A block structured grid was generated for comparison, shown in Fig. 5.13(b), and the same flow case was simulated using PMB. The surface pressure distribution is compared with that of the meshless computation, and the results are given in Fig. 5.13(d), which show good agreement.

	time (s)	%
Check boundary overlap	0.00001	<0.1
Point blanking	0.03	0.07
Stencil selection	1.62	3.86
Final boundary check	0.03	0.07
Solver time	40.23	95.76

Table 5.1: Timings for each stage of the preprocessor in seconds and as a percentage of total computational time for the subsonic biplane case.

The timings for each of the steps involved in the PML scheme are given in Table 5.1. Steps one, two and four of the preprocessor are the most cost effective, which is due to only the boundary elements being processed. Step three is the most expensive due to every point in the domain having to perform a global search, and then select the best stencils from the candidate points found. The preprocessor stage is about 4% of the total solution time. It should be noted that this is for a steady case, and so convergence



(a) Meshless point distribution

(b) Block structured grid used by PMB

(c) PML pressure contours

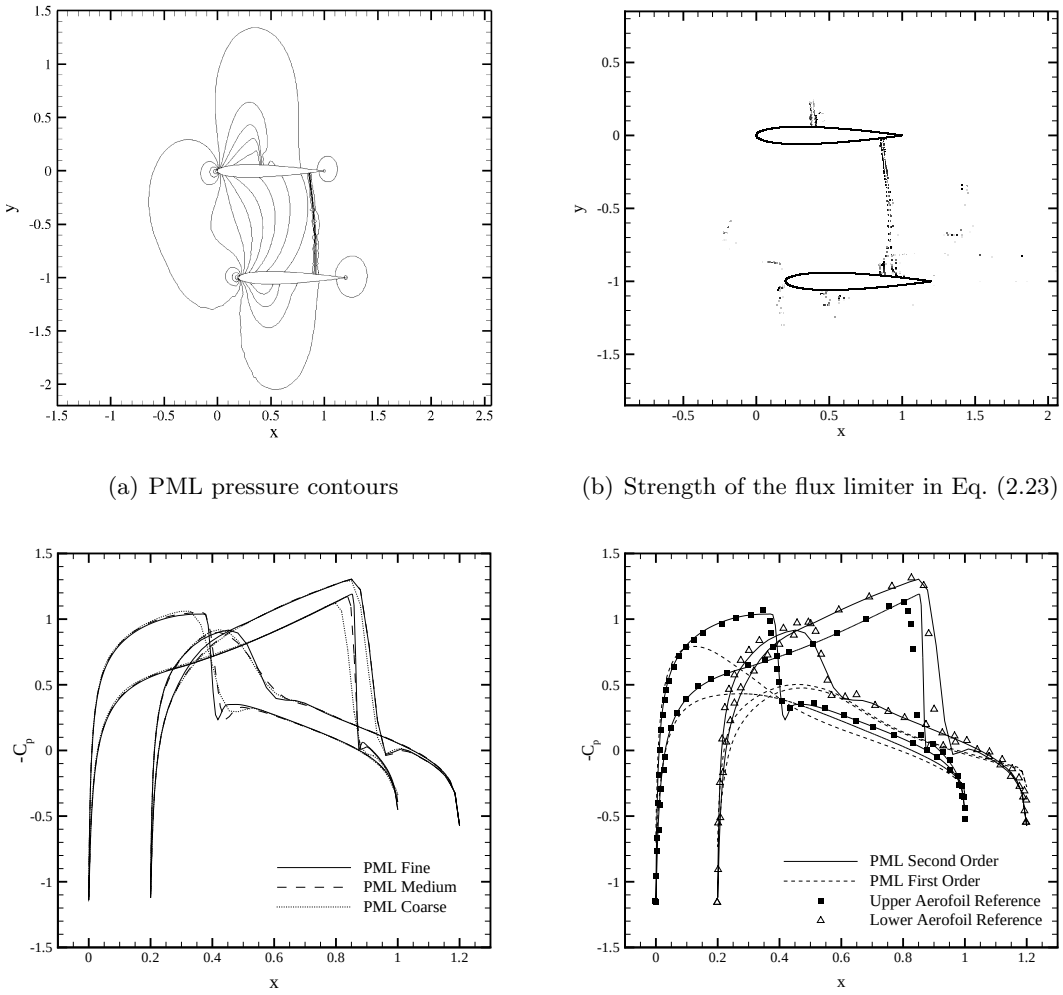
(d) Surface pressure distribution

Figure 5.13: Results for biplane configuration solving the Euler equations at M_∞=0.5, α=0° compared with a finite volume result using PMB.

is achieved to five orders of magnitude from the initial, freestream conditions. It also means that explicit steps are required as a start-up before the implicit solver is used, which takes up about 50%-60% of the solver time. The start up is not required for every step in an unsteady calculation (only the first), and convergence is to three orders of magnitude per time step; such cases will be seen in Section 6.2.2.

5.4.2 Transonic steady flow using input domains of varying density

The second case tests the stencil selection when the point distributions in the overlap region are of varying density. For this another biplane case is considered, with one aerofoil using the same point distribution, while the other is of increasing refinement. The meshes for the second aerofoil are labelled coarse, medium and fine, and are of size 5686 points, with 128 points on the solid wall; 22380 points, with 256 points on the



(a) PML pressure contours

(b) Strength of the flux limiter in Eq. (2.23)

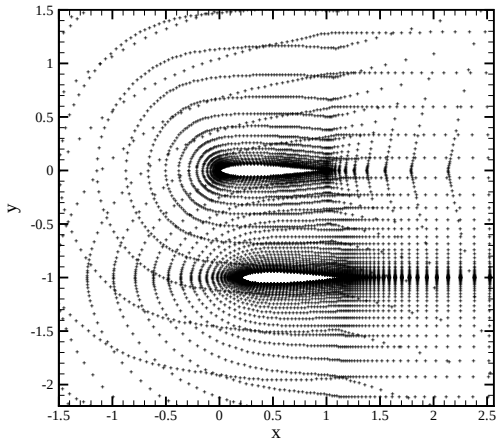
(c) PML pressure distribution

(d) Comparison with Ref. [134]

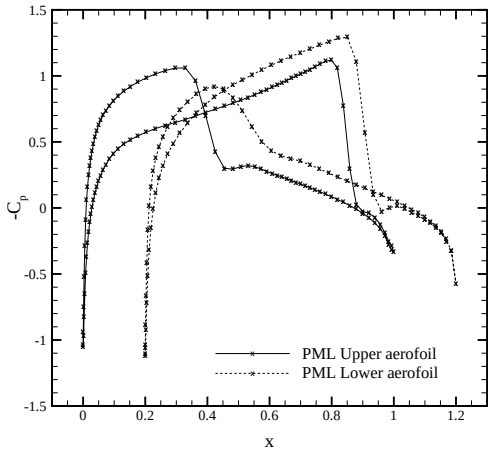
Figure 5.14: Results for biplane configuration at inviscid flow conditions M_∞=0.755, α=0.016° compared with Ref. [134], which uses a Chimera solver.

boundary wall; and 88792, with 512 on the solid wall. The first aerofoil uses the grid from Section 5.4.1 (so is the coarsest of all), with the leading edge located at (0.2,-1.0); the second aerofoils have the leading edge at the origin.

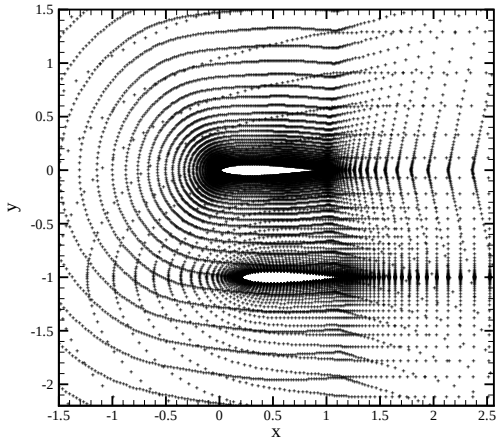
The flow conditions for this case have a freestream Mach number of 0.755 and an angle of attack of 0.016°, so there is a strong shock between the aerofoils within the overlap region; this can be seen in the pressure contours in Fig. 5.14(a). The surface pressure coefficients for each point distribution are plotted together in Fig. 5.14(c). It can be seen that the shock becomes better resolved on the upper aerofoil as the point distributions become more refined; the shock on the lower aerofoil is improved only slightly because this point distribution remains the same throughout. There are no errors caused by the point distributions being of different densities, which would be the case if there were significant differences in the solutions. Fig. 5.14(d) presents the



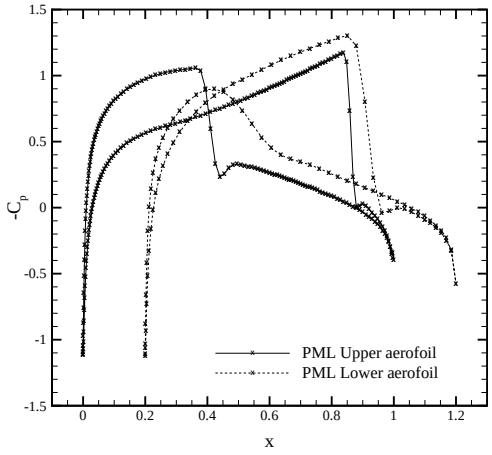
(a) Coarse point distribution



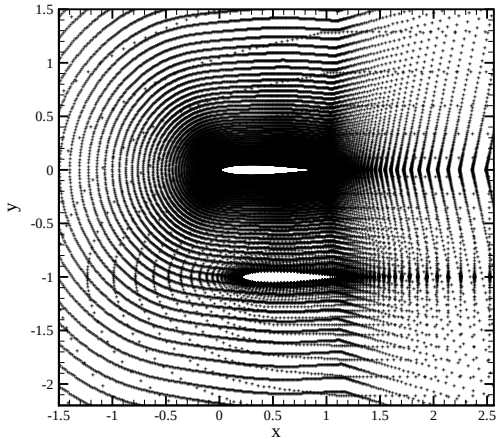
(b) Coarse surface pressure distribution



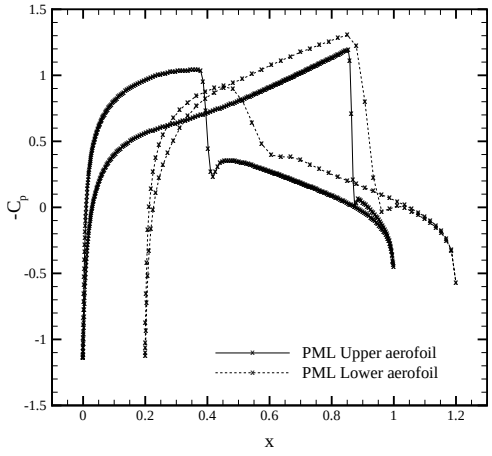
(c) Medium point distribution



(d) Medium surface pressure distribution

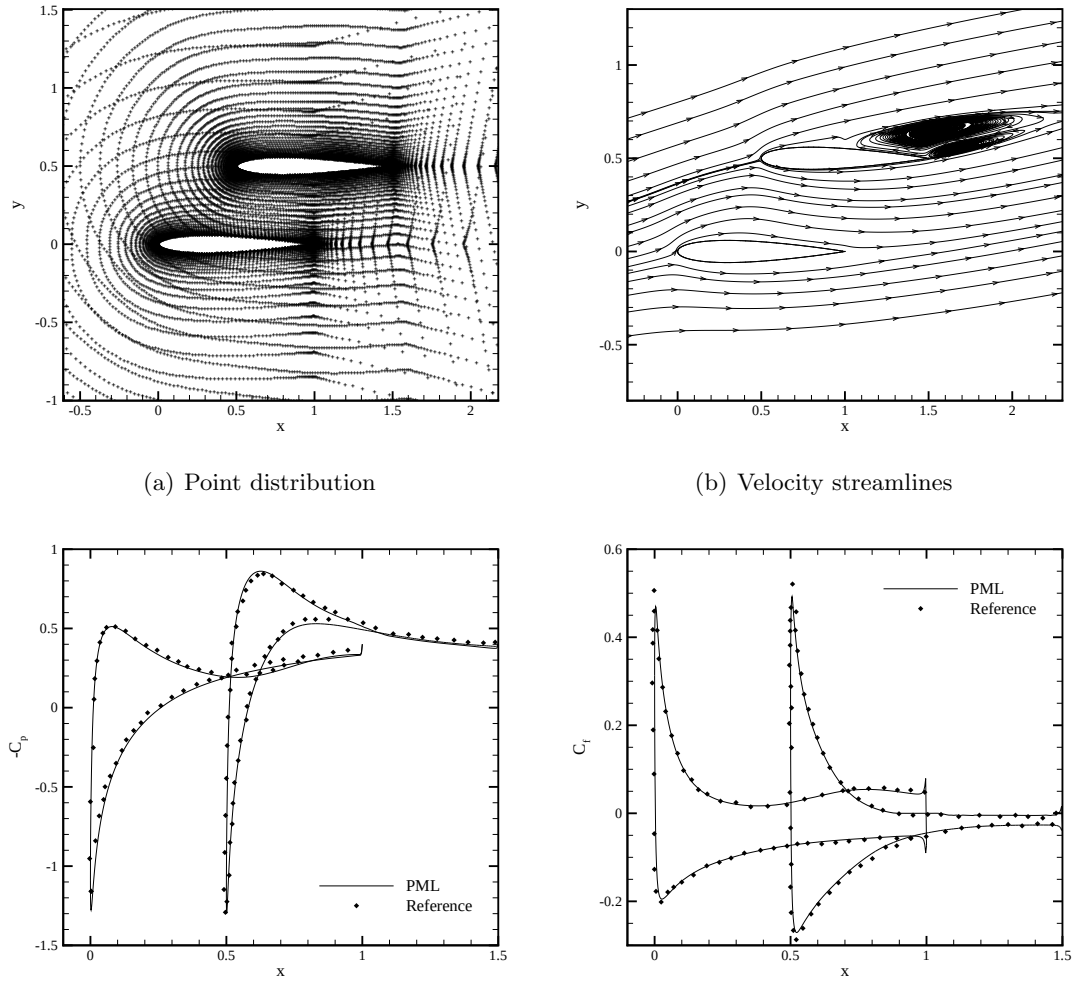


(e) Fine point distribution



(f) Fine surface pressure distribution

Figure 5.15: Results for biplane configuration at inviscid flow conditions of M_∞=0.755, α=0.016° with a coarse, medium and fine upper aerofoil point distribution.



(a) Point distribution

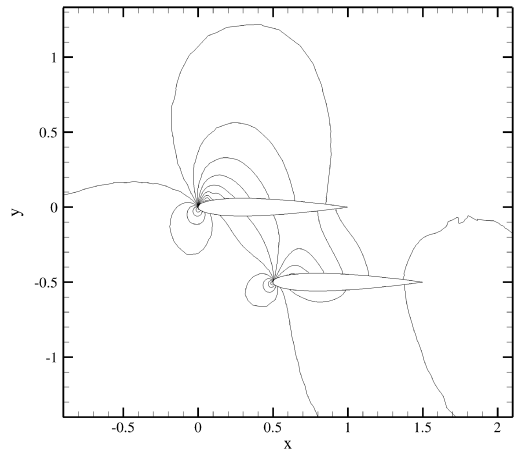
(b) Velocity streamlines

(c) Pressure coefficient

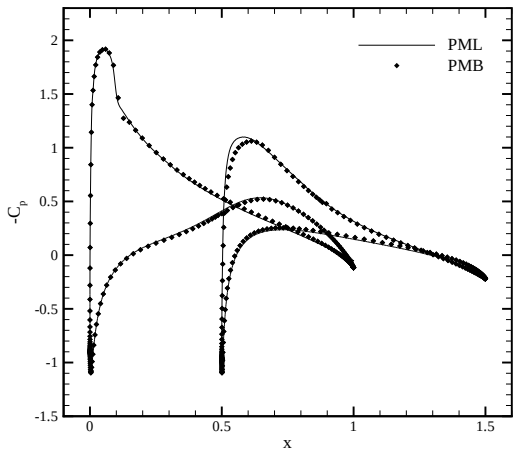
(d) Skin friction coefficient

Figure 5.16: Results for the laminar biplane case at M_∞=0.8, α=10° and Re=500 compared with numerical results from Ref. [135].

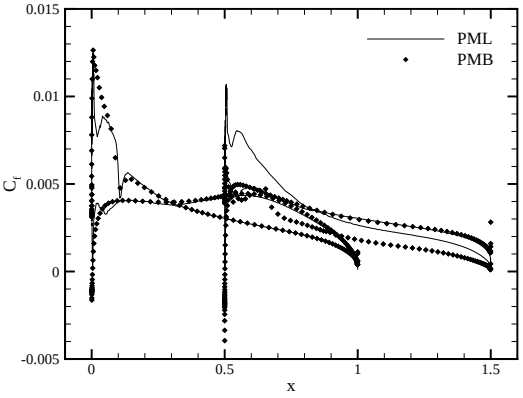
second order PML flow results for the case with the fine upper aerofoil, and numerical results from Ref. [134], which uses the chimera grid method on two overlapping aerofoils in a structured grid format. These results show good agreement, though there is a slight difference for the shock size on the lower aerofoil due to the coarse aerofoil used at this location. The first order results of PML are included to show the effect of the second order terms in Eq. (2.23). The second order accuracy can also be seen in Fig. 5.14(b), which plots where the flux limiter applied to the pressure has effect; it can be seen that these regions coincide with the shock locations as expected. The point distributions and surface pressure plots are shown in Fig. 5.15, in which we can see the increased refinement of the points of the upper aerofoil. Again we can see that the resolution of the shock waves is stronger as a result of the refinement, despite the mismatch of the overlapping grids in point density.



(a) PML Pressure contours



(b) Surface pressure distribution

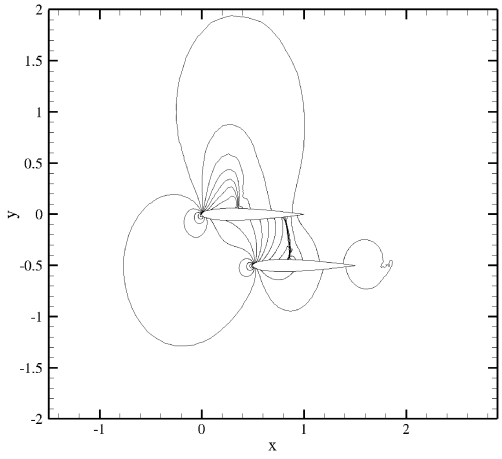


(c) Skin friction coefficient

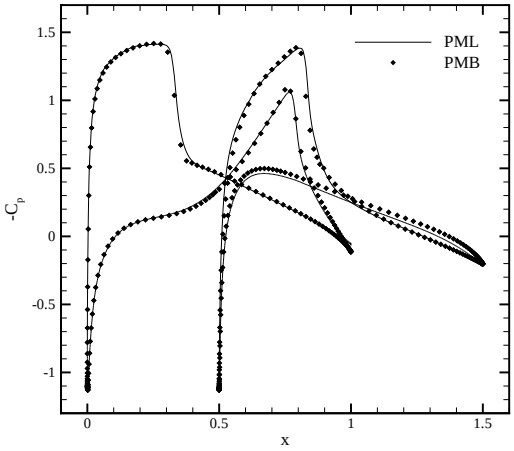
Figure 5.17: Results for the RANS biplane case at M_∞=0.6, α=2.89° and Re=4.8 million compared with finite volume results from PMB.

5.4.3 Laminar steady state flow

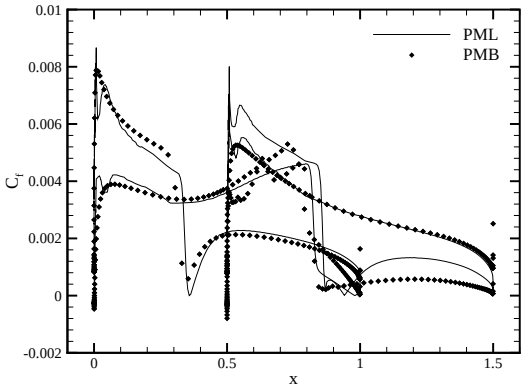
This test case involves simulating laminar flow past two NACA 0012 aerofoils, forming a staggered biplane so that the lower aerofoil has its leading edge at the origin, while the upper aerofoil has its leading edge at (0.5,0.5). The lower aerofoil consists of 15428 points, with 176 points on the solid wall, and the upper aerofoil is the same as that used for the medium point distribution for the test case in Section 5.4.2; thus, this case demonstrates the use of overlapping point distributions of different resolutions for viscous flows. The flow conditions are $M_\infty = 0.8$, $\alpha = 10°$ and $Re = 500$. A view of the resultant point distributions is given in Fig. 5.16(a); it can be seen that points from one input domain will fall into the region close to the boundary walls of the other input domain. The streamlines shown in Fig. 5.16(b) show two vortices; the secondary vortex is due to the presence of the lower aerofoil as this effect is not seen for a single aerofoil under these flow conditions, as is discussed in Ref. [135]. This reference also presents pressure and skin friction coefficient distributions for this case; the results from PML are in good agreement with this data, as can been seen in Figs. 5.16(c) and 5.16(d).



(a) PML Pressure contours



(b) Surface pressure distribution

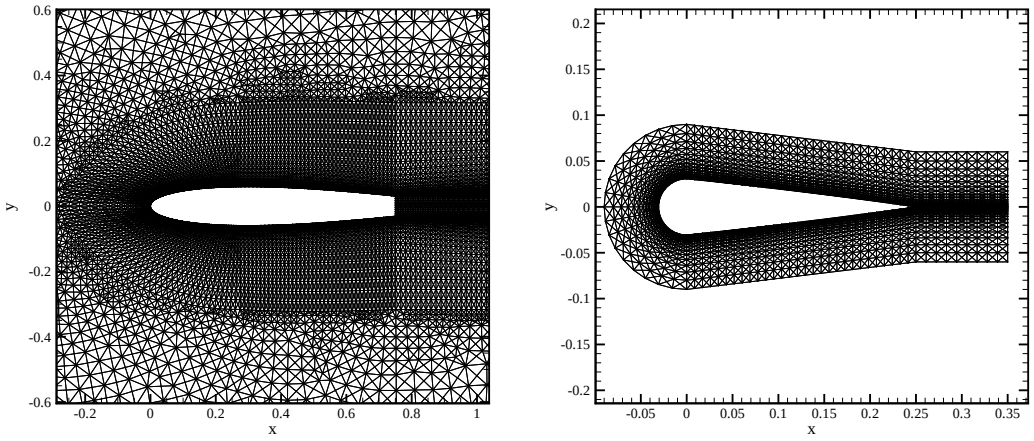


(c) Skin friction coefficient

Figure 5.18: Results for the RANS biplane case at M_∞=0.7, α=1.49° and *Re*=9 million compared with finite volume results from PMB.

5.4.4 RANS steady state flow

The biplane configuration is further analysed using the preprocessor and meshless method, for solving the RANS equations with the SA turbulence model. The aerofoils are two NACA 0012 aerofoils of 15017 points, with 308 of those on the solid wall, for each grid. The locations of the aerofoils are the same as in Section 5.4.1; and it should be noted that the point distributions have the same form as the aerofoil in Fig. 4.1(b), that is, they have an anisotropic, structured format near the boundary walls and an unstructured format further away. The minimum wall normal spacing for the aerofoils is 10^{-5} of the chord length. The flow conditions are at $M_\infty = 0.6$, $\alpha = 2.89°$ and $Re = 4.8$ million; coinciding with the flow conditions for the AGARD CT1 case for a single aerofoil, which was studied in Section 4.1.6. The pressure contours for this case are shown in Fig. 5.17(a). This case is also compared with the flow results from PMB, for which a structured grid, of the same format as in Fig. 5.13(b), was generated; and the surface pressure distributions and skin friction coefficient are shown in Figs. 5.17(b) and 5.17(c) respectively, in which good agreement is obtained.



(a) Main body connectivity

(b) Control surface connectivity

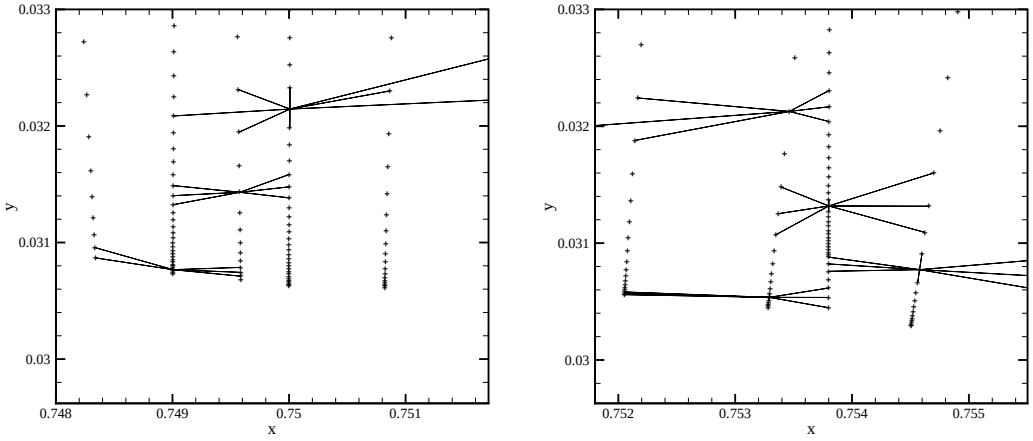
Figure 5.19: Input connectivities of the main body and control surface geometries.

A transonic case is also tested, with input domains of 36437 points, with 464 points on the solid wall, in the exact same locations. The flow conditions are set to $M_\infty = 0.7$, $\alpha = 1.49°$ and $Re = 9$ million; these conditions produce two shocks formed on the upper aerofoil and one on the lower, which can be seen in Fig. 5.18(a). The same case is run with PMB as a comparison, and the surface pressure distribution and skin friction coefficient is given in Figs. 5.18(b) and 5.18(c) respectively, which again show good agreement.

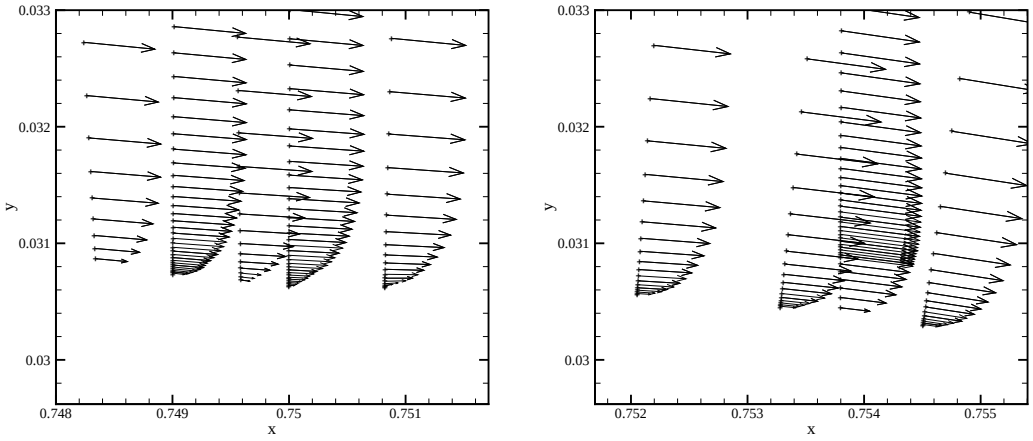
5.4.5 Deflected control surface case

In this section, steady state simulations of an idealised control surface configuration at various angles of deflection are performed. It is possible to generate a different point distribution to accommodate each deflection separately, though the flexibility of the preprocessor and meshless method is employed by overlapping individual component grids representing the main body, which consists of 94849 points, with 415 on the solid wall, Fig. 5.19(a); and the control surface, which is a much smaller domain consisting of 13360 points, with 256 on the solid wall, Fig. 5.19(b). These components are used to solve the RANS equations with the SA turbulence model; and each have a minimum wall normal point spacing of 1×10^{-5}.

Note that the main body is a truncated NACA 0012 aerofoil with a blunt trailing edge at $x = 0.75$ ($0.75c$ of the original aerofoil); the point distribution has an anisotropic, structured format near the boundary walls, which changes to an unstructured format further away (about $0.3c$ of the original aerofoil) from the solid wall. The control surface is a component mesh, in a structured format, with its origin positioned at location (0.75, 0), overlapping the main body point distribution; thus, the boundaries of the input domains intersect. The control surface body rotates around this



(a) View of some stencils at intersection location (b) View of stencils along control surface location

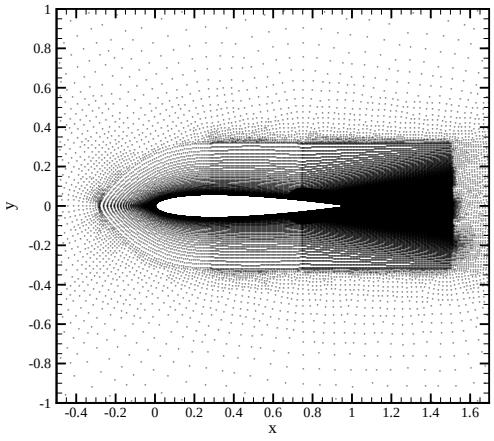


(c) Velocity vectors at intersection location (d) Velocity vectors along control surface location

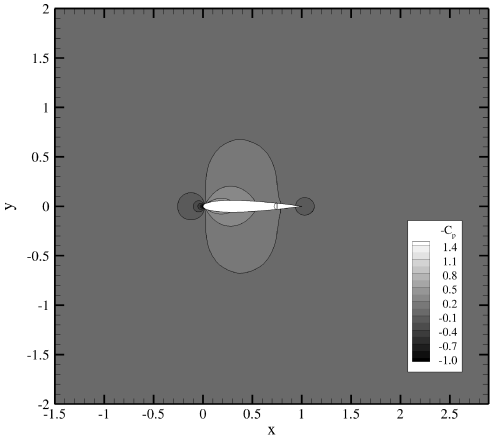
Figure 5.20: Close-up of boundary layer region at the intersection between the two bodies on the upper surface at β=20° at turbulent flow conditions Mach 0.2, zero degrees angle of attack and Reynolds number 5 million.

position to provide the required deflection at angles, β, of 0°, 10° and 20°, where the positive angle represents a downward deflection. The equations are solved at each of these configurations, at flow conditions $M_\infty = 0.2$, $\alpha = 0°$ and $Re = 5$ million, to demonstrate the capabilities of the preprocessor and flow solver for such cases.

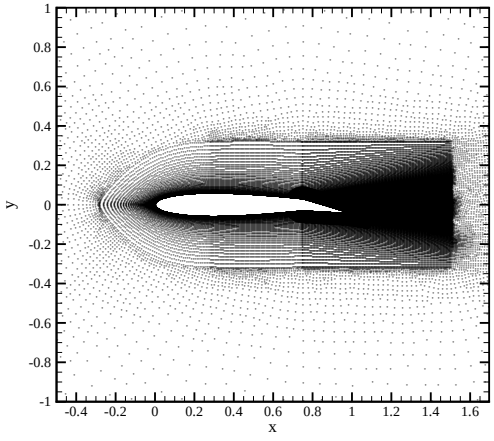
Figure 5.20 shows close-ups of the upper surface when the control surface is at angle $\beta = 20°$, at locations near the intersection point between the bodies and just along the surface of the control surface. The set of points at $x = 0.749$ and $x = 0.75$ in Figs. 5.20(a) and 5.20(c) belong to the main body, while the remaining points belong to the control surface; it is clear that points from the control surface fall into the boundary layer region of the main body. Similarly, points from the main body fall into the boundary layer region of the control surface in Figs. 5.20(b) and 5.20(d): the points at $x = 0.7538$ belong to the main body, while the rest belong to the control



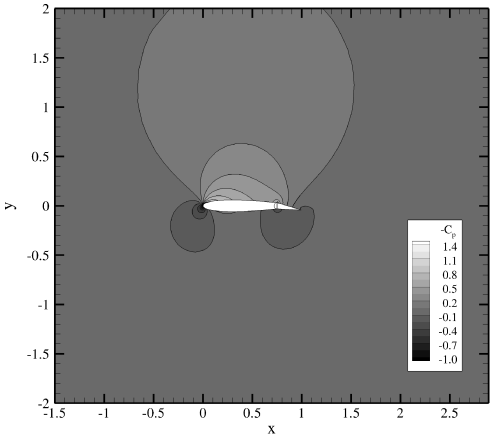
(a) Point distribution at $\beta = 0°$



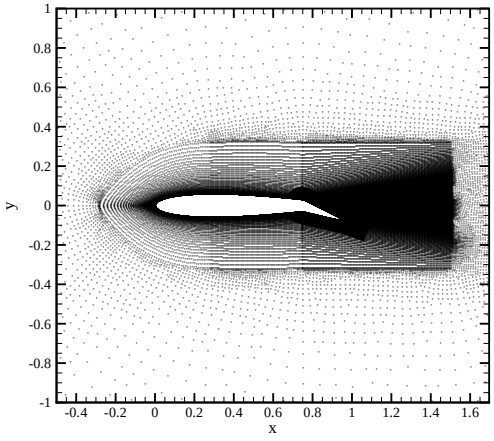
(b) Pressure coefficient plot at $\beta = 0°$



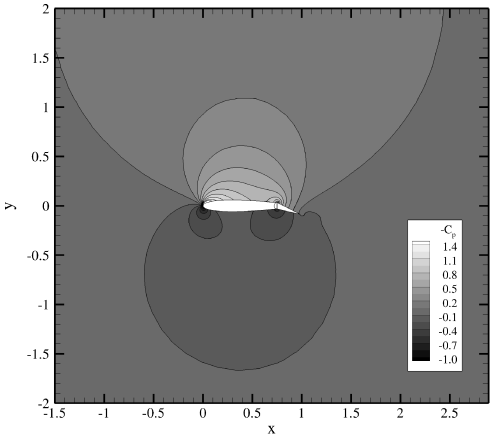
(c) Point distribution at $\beta = 10°$



(d) Pressure coefficient plot at $\beta = 10°$



(e) Point distribution at $\beta = 20°$



(f) Pressure coefficient plot at $\beta = 20°$

Figure 5.21: Point distribution of the main body and control surface configuration at various angles of deflection β, with pressure coefficient plots at turbulent flow conditions M_∞=0.2, α=0°, Re=5 million.

surface. Figures 5.20(a) and 5.20(b) show some of the stencils in these regions; all of the stencils shown are anisotropic and contain points belonging to both bodies. It is interesting to note that the stencils that are located second furthest from the wall in Fig. 5.20(a), and furthest from the wall in Fig. 5.20(b) contain only six points. This is because the ψ_{max} condition is violated for the points that are located directly above and below the star points of the stencils in question; both of these points belong to the point distribution generated around the control surface, and the set of points from the boundary wall that the stars belong to are coarser than the sets either side in these locations, which is why this occurs. The velocity vectors are plotted for these points in Figs. 5.20(c) and 5.20(d), from which the boundary layer can be distinguished.

The resultant point distributions of the overlap at β of 0°, 10° and 20° are presented in Figs. 5.21(a), 5.21(c) and 5.21(e) respectively; and the pressure coefficient plots of each of these cases is presented in Figs. 5.21(b), 5.21(d) and 5.21(f) respectively. It can be seen that for the case $\beta = 0^\circ$ the configuration acts just as a NACA 0012 aerofoil, at the given flow conditions. At $\beta = 10^\circ$ and $\beta = 20^\circ$ the camber increases, causing an unsymmetric flow over the configuration, which would produce a pitch down motion. Control surface deflections are analysed further in Chapter 6 for unsteady, moving-body flows.

5.5 Preprocessor in three-dimensions

The development of the preprocessor in three-dimensions follows the same reasoning as for the two-dimensional preprocessor outlined in Section 5.1; and so the extension to three-dimensions is relatively straightforward. However, there are more complications with the method in three-dimensions, particularly regarding the computational geometry required in the boundary reallocation and point blanking operations. For example, in two-dimensions two infinitely long lines will always intersect unless they are parallel; in three-dimensions this is not necessarily the case, as it is possible for two lines to be skew. This can cause difficulties regarding tolerances when determining intersections; consequently, projections must often be made onto two-dimensional surfaces in such operations for robustness. Due to complications such as this, it is worth summarising the procedure again separately for clarity; some of the details referred to are outlined in Appendix C. The input required and steps making up the procedure for the three-dimensional preprocessor are the same as described in Section 5.2. The form of the initial, input stencil connectivities for interior and boundary point stencils can be seen in Fig. 4.12 for reference; and each of the steps required is outlined in the following sections.

5.5.1 Redefining boundaries

The first step of the preprocessor is to see if the boundary elements of the composite domains intersect. The initial test for boundary overlap in three-dimensions follows the same procedure as for the two-dimensional method in Section 5.3.1. First, bounding boxes are formed around each element, as in Fig. 5.22. A search tree is created for each input domain, containing all of the boundary elements from the respective domain. The trees store the coordinate limits of the bounding box

$$\mathbf{x} = \{x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}\}$$

so each element can be represented as a point in six-dimensions.

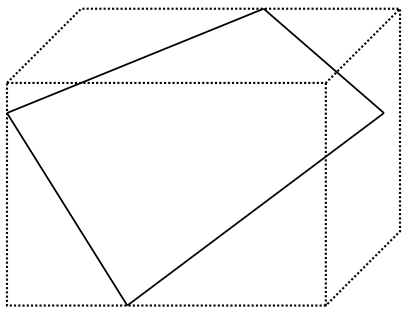
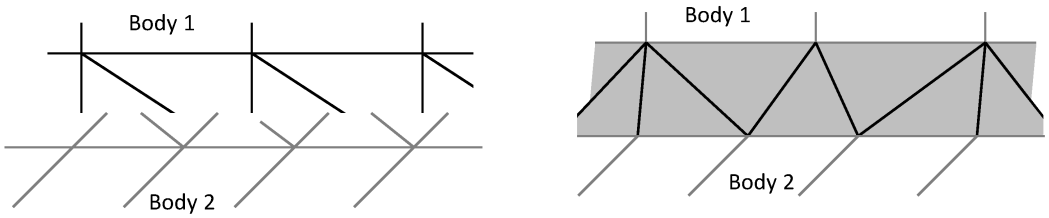


Figure 5.22: Bounding box around quadrilateral element.

We test for element intersection by traversing each search tree, excluding the tree to which the element belongs, using the bounding box of the element in question as the search region. If elements are found in the tree, using the six-dimensional form of Eq. (5.1), then the bounding boxes of the elements found intersect that of the element in question; though, of course, we still need to explicitly check if the elements within the boxes intersect.



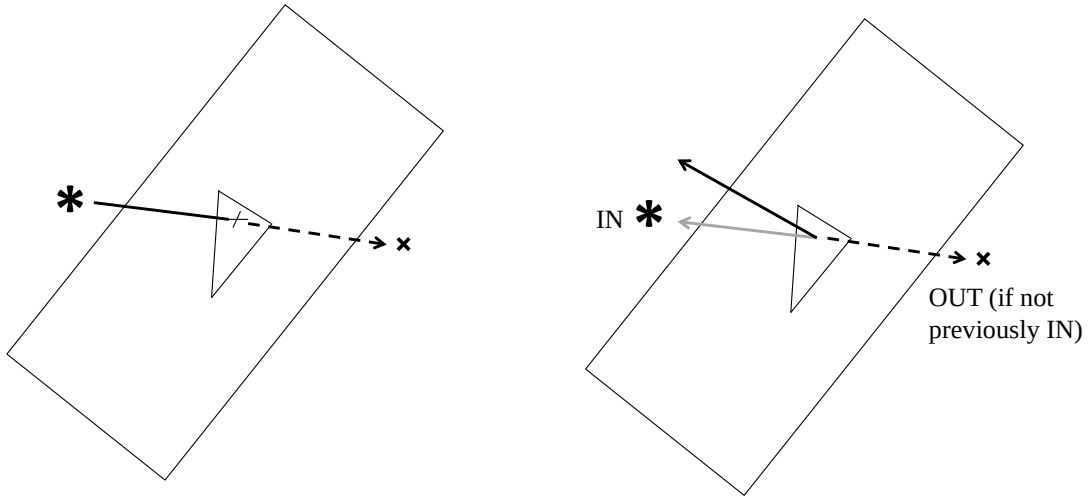
(a) Two boundaries from different bodies intersecting

(b) Successful triangulation across the boundary intersection

Figure 5.23: Method of redefining the solid boundaries after intersection.

The test for element intersection is more difficult than the segment intersection performed in two-dimensions. For example, complications arise if the elements that make up the boundary surfaces have more than three sides. This means that the points making up each element are not necessarily coplanar, and so operations using computational geometry are more difficult to perform robustly; as such, it is necessary to break the elements up, even if only temporarily, into triangles. As the points making up a triangle *do* share the same plane, they have a well defined normal for which the algorithms that are used are robust; though this means that if the elements are quadrilaterals, more element intersection tests must be made per bounding box intersection. The algorithm to test for an intersection between two triangles T_1 and T_2 in a three-dimensional space can be found in Ref. [136]; the principles of the method are stated in Appendix C.2 briefly for completeness. Difficulties arise with the algorithm if the triangles to be tested are nearly coplanar, or when an edge is nearly coplanar to the other triangle: see the reference for more information on these cases.

If the triangles intersect, then the elements of the geometry intersect. If the boundary elements belong to solid walls, then the boundaries are reallocated to account for the intersection; if any other combination of boundaries intersect, which may occur in an unsteady, moving-body simulation because of the relative motion due to the forces acting on the bodies, then an error is returned and the preprocessor stops. New elements are created over the intersection region to account for the intersection of solid bodies, which is a preferred method over creating new points, for the same reasons as stated in Section 5.3.1 for the preprocessor in two-dimensions. The new elements are triangles, attempting to form a Delaunay triangulation, formed from the edge of an intersecting element and a point from the intersecting body: so that at least one of the points of the new triangles comes from each of the bodies that intersect, Fig. 5.23. The ability of the flow solver to operate on surfaces consisting of quadrilateral or triangular elements was demonstrated in Section 4.2.1. To accomplish this boundary reallocation, it is necessary to flag each element for which an intersection occurs, and also to store the elements that intersect it. The subsequent procedure used is written in six steps for brevity in Appendix C.3. The ideal result of the operation is shown in Fig. 5.23, and this will be seen in some of the examples shown in Section 5.6.



(a) Ray between star point and stencil point intersects the triangle within the plane

(b) Normal test identifies the star as IN, the stencil point may be OUT

Figure 5.24: Blanking point operations in three-dimensions.

5.5.2 Blanking points

The general procedure for blanking points in three-dimensions is very similar to that in two-dimensions, except that the possible intersection of an initial stencil and boundary element is identified using the six-dimensional search tree algorithm outlined in the previous section, and the geometric intersection uses a ray-element intersection algorithm. If the bounding box around the initial, input stencil and the bounding box around the element intersect, then a ray is formed between the star point and every other point in the stencil. Each ray is tested against the element found for an intersection, to see if either point may need to be blanked. Again, if the element is a quadrilateral then it must first be broken up into triangles. The algorithm for a ray-triangle intersection, which is also used in the reallocation of the boundaries, is stated in two steps in Appendix C.1. An intersection means that either the star or neighbouring point may lie within a boundary wall, and so both must be checked to see if either should be blanked, Fig. 5.24. The same normal tests described in Section 5.3.2 are used to classify the points; and the same flood operation is used at the end to blank remaining points internal to a solid wall.

5.5.3 Stencil selection

The concept of resolving vectors is continued to the stencil selection in three-dimensions. Individual, local resolving vectors are again determined from calculating the principal axes of inertia of the point cluster, where each point has unit mass. In three-dimensions these vectors can be found from calculating the eigenvectors of the moment of intertia

tensor (denoted I, but not to be confused with the identity matrix), written as

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix}$$

where the elements of this matrix are the products of inertia, given by

$$\begin{aligned} I_{xx} &= \sum_{j \in \Omega_i} \{(y_j - y_i) + (z_j - z_i)\} \\ I_{yy} &= \sum_{j \in \Omega_i} \{(x_j - x_i) + (z_j - z_i)\} \\ I_{zz} &= \sum_{j \in \Omega_i} \{(x_j - x_i) + (y_j - y_i)\} \\ I_{xy} &= -\sum_{j \in \Omega_i} (x_j - x_i)(y_j - y_i) \\ I_{xz} &= -\sum_{j \in \Omega_i} (x_j - x_i)(z_j - z_i) \\ I_{yz} &= -\sum_{j \in \Omega_i} (y_j - y_i)(z_j - z_i) \end{aligned}$$

The eigenvectors of I can be found numerically using the Jacobi iterative method. To determine which of these vectors should be the local resolving vector, we perform a change of basis so that the eigenvectors form the new coordinate system of the initial stencil, Fig. 5.25, with the star point located at the origin. In this way the x, y and z axes coincide with the eigenvectors found. We then find the largest absolute value of x, y and z, labelled $|d_x|$, $|d_y|$ and $|d_z|$ respectively, of *all* of the points in this basis, and the smallest of these maximum values determines the choice of eigenvector to be the resolving vector; so,

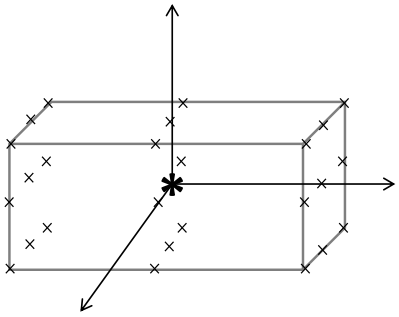
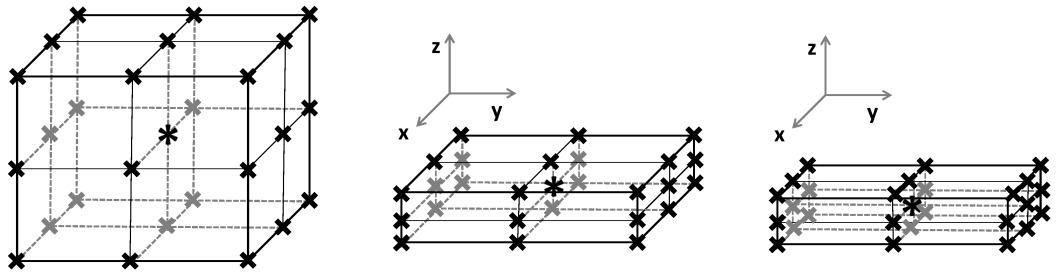


Figure 5.25: Determining the resolving vector using the principal axes of inertia.

$$\mathbf{v} = \begin{cases} \mathbf{v}_{eig,x}, & \text{if } |d_x| = min(|d_x|, |d_y|, |d_z|) \\ \mathbf{v}_{eig,y}, & \text{if } |d_y| = min(|d_x|, |d_y|, |d_z|) \\ \mathbf{v}_{eig,z}, & \text{if } |d_z| = min(|d_x|, |d_y|, |d_z|) \end{cases}$$

where $\mathbf{v}$ is the resolving vector choice. The length of the resolving vector is set according to the smallest length, denoted γ, in the initial stencil, in the same way as Eq. (5.4); and the value $p = 4$ is retained.



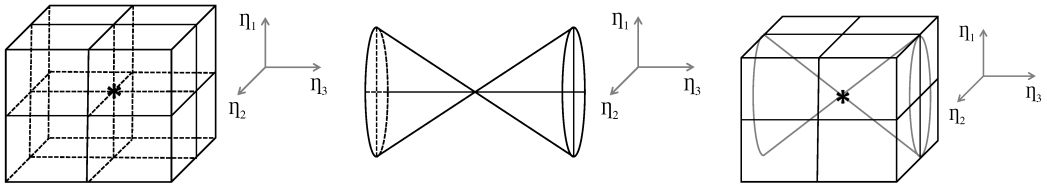
(a) Stencil with no anisotropy (b) Stencil with one direction of anisotropy (c) Stencil with two directions of anisotropy

Figure 5.26: Directions of anisotropy for stencils in three-dimensions.

Recall, from Section 5.3.3, that the resultant resolving direction defines a new coordinate system, in which the merit function is applied to a list of candidate points. In two-dimensions, two vectors are needed to define a basis: one is the resolving vector and the other is the vector orthogonal to this; however, in three-dimensions, three vectors are needed to define a basis: one is the resolving vector, but the other two vectors could be any orthogonal vectors within the plane that is normal to the resolving vector. Consider the various stencil types in three-dimensions that reflect the various directions of anisotropy, shown in Fig. 5.26. The stencil in Fig. 5.26(a) is isotropic, so the vectors are trivial; the stencil in Fig. 5.26(b) is fine only in the z direction, so the remaining two vectors could lie anywhere in the xy plane; the stencil in Fig. 5.26(c) is fine most in the z direction, but it is also fine in the x direction, so the choice of basis in which to define the coordinate system is more constrained and should reflect this point direction. Thus, the choice of vectors is not trivial, as stencils can be anisotropic in two directions in three-dimensions: it is therefore necessary to introduce another vector to resolve the functions. This vector is orthogonal to the vector defined above, and is called the orthogonal resolving vector $\mathbf{v}_o$; and points in the direction where there is second most refinement: the orthogonal resolving vector for the case in Fig. 5.26(c), therefore, points along the x axis. The length of this vector is determined by the smallest length, denoted γ_l, in the initial stencil along this axis, with which the vector varies inversely, thus,

$$\mathbf{v}_o = \frac{\hat{\mathbf{v}}_o}{\gamma_l^p} \tag{5.5}$$

When the overlapping stencils are identified, using the same procedure described in Section 5.3.3, the sum of resolving vectors *and* orthogonal resolving vectors is made. The sum of the resolving vectors defines one axis of the new coordinate basis $\boldsymbol{\eta}$; the second axis is taken as that within the plane normal to the resolving vector, which forms the smallest angle with the sum of the orthogonal resolving vectors. It is important that the resolving vectors to be summed point in the same direction (that is, the angle between them is less than 90°), otherwise the vectors will have the effect of cancelling out, not summing: this is checked before the sum takes place. It is easy to reverse the



(a) Eight sectors surrounding the star point

(b) Additional cones that will be placed amongst sectors

(c) Union of sectors

Figure 5.27: Selection of points in basis η for three-dimensional point distributions.

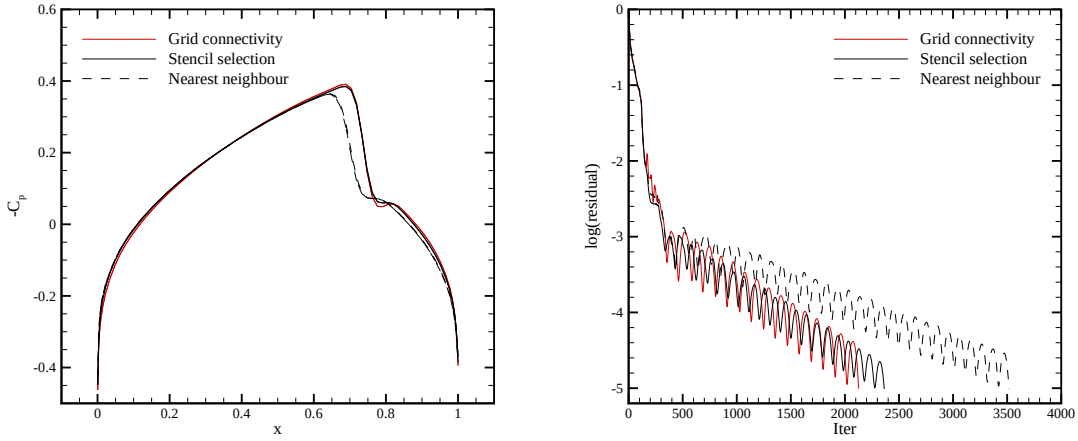
direction of one of the vectors for the summing process if this is the case. The third vector in the basis is, of course, orthogonal to the first two vectors, and can be found using the vector product. It is in this basis that the merit function is applied.

The ellipse shaped merit function used in two-dimensions is replaced by an ellipsoid merit function, which will rate each candidate point in terms of the required direction and refinement by balancing the orthogonality of the points chosen (for refinement) and the distance. Setting the star to be at the origin, we define ξ_1, ξ_2 and ξ_3 to be the coefficients of η respectively, that form the coordinates of each candidate point in the basis. The merit function ψ takes the form

$$\psi = \frac{\xi_1^2}{a^2} + \frac{\xi_2^2}{b^2} + \frac{\xi_3^2}{c^2}$$

The constants a, b and c are chosen from projections onto the axes in the space $\boldsymbol{\eta}$, as in the two-dimensional scheme. The points are selected in the basis according to octants to keep the stencils well-conditioned; these octants are positioned along the $\eta_1\eta_2$, $\eta_2\eta_3$ and $\eta_1\eta_3$ planes, Fig. 5.27(a). Furthermore, there are additional cones introduced to impose the ψ_{max} condition of Section 5.3.3, and to take into account the second direction of anisotropy; these are placed with the opening angle pointing in the direction of *l*east refinement, as shown in Fig. 5.27(b), and are used in conjuction with the octants to form the full cofiguration of 16 sectors, Fig. 5.27(c).

To form an overdetermined, linear system in three-dimensions we need more than four points, but due to stability issues we require more. To be consistent with the stencils forming the input domains, which are also used in Chapter 4 and are obtained from structured grids, the stencils are constructed so that they contain 27 points. First, one point is chosen from each cone with the smallest value of ψ; then two points with the lowest values of ψ are chosen from each of the octants within the cones to form part of the stencil. The maximum value of ψ from these 18 points is denoted ψ_{max}. To retain accuracy we then take one point from each of the octants that are not within the cones, each with the lowest value of ψ, provided these values are less than ψ_{max}. Thus, in this scheme, the stencils for three-dimensional meshless calculations have between 18



(a) Pressure coefficient at $y/y_{tip} = 0.15$

(b) Convergence history

Figure 5.28: Comparison of PML flow solution and convergence history using initial grid connectivity and stencil selection methods on a Goland wing at freestream Mach number 0.9 and 0 degrees angle of attack.

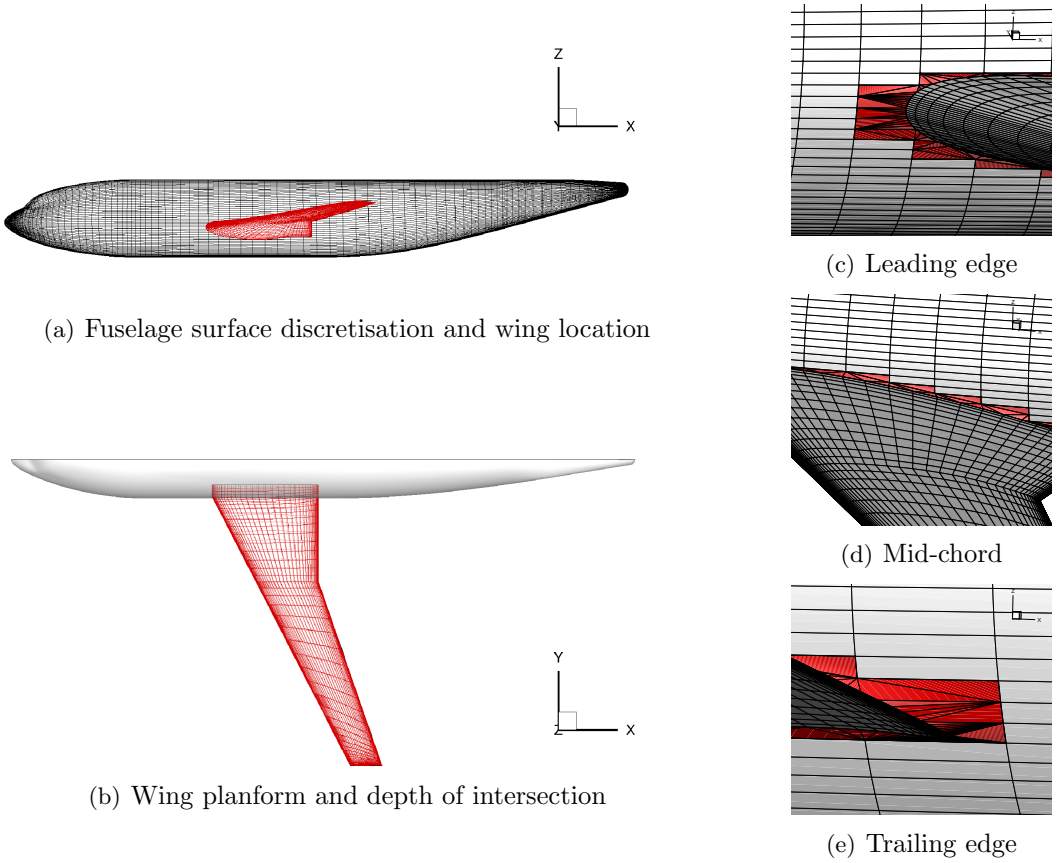
and 26 points in, not including the star point itself. Larger stencils can be constructed, by simply selecting more points from the sectors, if needed for higher order polynomial reconstruction. Such large stencils ensure not only that the least squares system is over-determined, but also that there is sufficient overlap of the resultant stencils to produce accurate, well converged solutions.

The final boundary check, as is used in Section 5.3.4, is needed to make sure that the selected stencils conform to the boundaries. The method is the same, but with the ray-element intersection, described in the previous section, used to check the stencils.

5.6 Results in three-dimensions

In this section three test cases that demonstrate the capabilities of the preprocessor and solver are presented. For each case the preprocessor first works to select the stencils from various input geometries, then the meshless solver solves the Euler equations on the resultant domains. As in Section 5.4, the test cases do not reflect a real life aerospace application, but are designed to demonstrate the scheme.

The first case concerns the stencil selection on a single geometry; and comparisons are made with the flow solution when the grid connectivity of the input grid is used, and when a classical nearest neighbour approach is used. The second case is the DLR-F6 [137, 138] transport aircraft configuration. This case uses two input geometries: one around the wing and one around the fuselage; it therefore tests the ability of the scheme to work on intersecting geometries. The third and final case is a generic fighter aircraft configuration, based on publicly available data for the F-16; and consists of five intersecting input geometries to further demonstrate the method.



(a) Fuselage surface discretisation and wing location

(b) Wing planform and depth of intersection

(c) Leading edge

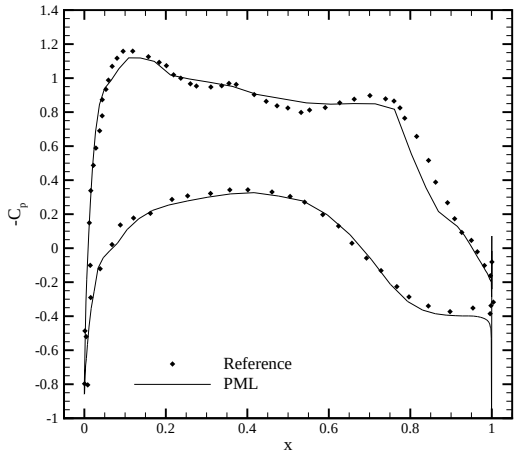
(d) Mid-chord

(e) Trailing edge

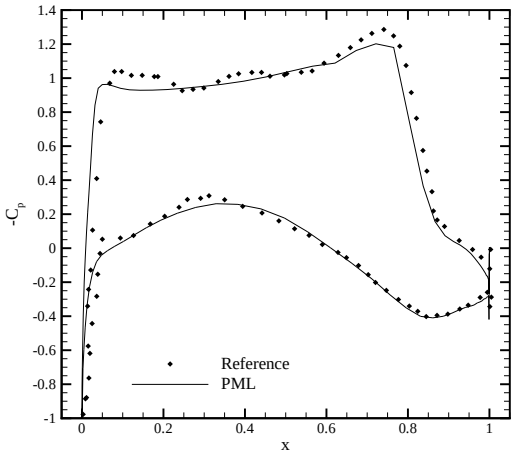
Figure 5.29: Configuration of the fuselage and wing input geometries, and new elements that are created by the preprocessor as a result of the geometries intersecting for the DLR-F6 test case.

5.6.1 Single geometry stencil selection

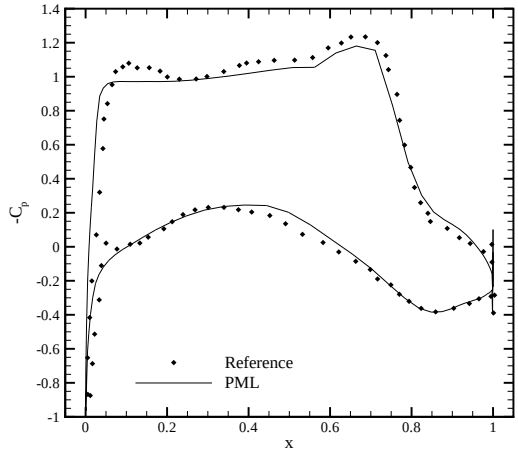
The method of calculating the local resolving vectors, evaluation of the parameters in the merit function and the selection of the points, was tested on a single geometry, so that the coupling of the resolving vectors and separate point distributions has no effect. The point distribution and initial connectivity used is the Goland wing case, which was tested in Section 4.2.1. The initial grid stencils are used to determine the resolving vectors and values in ψ; they are *not* used any further, as in keeping with the method when it is to be used on fully overlapping point distributions. The results of the flow solver using these stencils is compared with the results of the flow solver using the initial grid connectivity in Fig. 5.28, at inviscid flow conditions of $M_\infty = 0.9$ and $\alpha = 0$ degrees. It can be seen that the surface pressure distributions in Fig. 5.28(a) show good agreement; and the convergence rates in Fig. 5.28(b) are comparable. For comparison, results for this case are also presented for a nearest neighbour scheme using no resolving vectors and an octant search, with three points selected in each. The results show that even for a single point distribution, generated from a mesh, such a simple technique gives poor results.



(a) Spanwise location $\eta = 0.25$



(b) Spanwise location $\eta = 0.50$



(c) Spanwise location $\eta = 0.75$

Figure 5.30: Surface pressure coefficient plots for the DLR-F6 test case at M_∞=0.8 and α=0°, compared with results from Ref. [139].

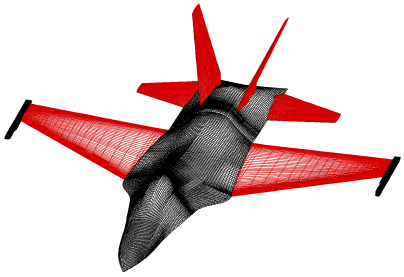
5.6.2 DLR-F6 fuselage-wing configuration

To continue the study, a more complex case is chosen for testing: the DLR-F6 configuration. The DLR-F6 represents a twin-engine, wide-body transport aircraft and has been the focus of several wind tunnel tests and computational studies. The test case presented consists of a simplified wing-fuselage geometry that has been used in the past for validation of CFD codes at the second [137] and third [138] AIAA sponsored Drag Prediction Workshops.

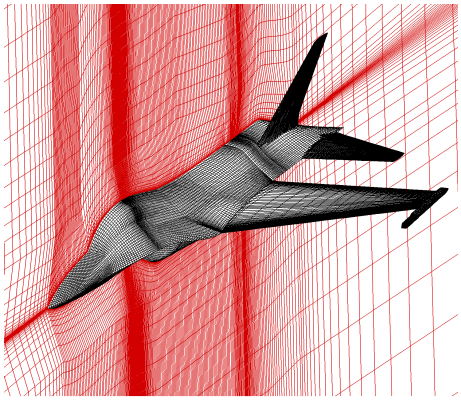
The procedure is tested by forming the F6 geometry from individual components for a steady state simulation. This is done from two input geometries: the fuselage, consisting of 2.5 million points, with 7098 on the solid wall; and the wing, consisting of 0.6 million points with 6730 on the solid wall. The location of the bodies can be seen in Figs. 5.29(a) and 5.29(b). The fuselage and wing intersect one another directly; and the new elements, created to accommodate the intersection, which are triangles as explained in Section 5.5.1, can be seen in red at the leading edge, mid-chord and

	time (s)	%
Boundary reallocation	1.33	0.14
Point blanking	5.88	0.63
Point searches	283.53	30.25
Point sorting	476.59	50.85
Boundary stencils	108.26	11.55
Final boundary check	61.70	6.58

Table 5.2: Timings for each stage of the preprocessor in seconds and as a percentage of total preprocessor time for the DLR-F6 case.



(a) View of the locations of the separate input domains.



(b) Solid wall surfaces (in black) and symmetry plane (in red).

Figure 5.31: Geometry of the OSF Case 2 when components are assembled.

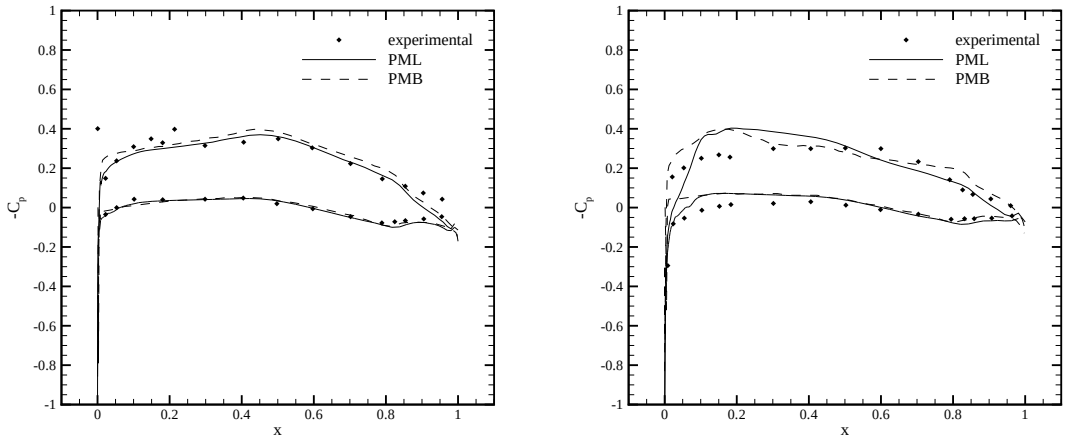
trailing edge location of the wing in Figs. 5.29(c), 5.29(d) and 5.29(e) respectively.

The timings for each stage of the preprocessor are presented in Table 5.2 to give a breakdown of the costs. As is the case for the two-dimensional scheme, the stencil selection stage is the most expensive, which consists of the point searching and point sorting stages. The selection of points for the boundary stencils is also part of the stencil selection: it is needed for the boundary points that form new boundary elements. The searches take nearly five minutes, the point sorting takes nearly eight minutes and the selection of the stencils for the required boundary points takes almost two minutes: this is about 92% of the cost of the entire scheme. Thus, future work on code efficiency should be made on these steps first.

The resultant stencils are used by the meshless flow solver to solve the Euler equations at Mach 0.8 and zero degrees angle of attack. The results for this case are shown in Fig. 5.30. Surface pressure coefficient plots at three spanwise locations along the wing are given in Figs 5.30(a), 5.30(b) and 5.30(c). These plots are compared with results from Ref. [139], which uses the complete finite volume grid of this configuration, and an unstructured flow solver at the same flow conditions. The results show good agreement for this case, and the shock location is well predicted.

	No. surface points	No. total points	Quantity	Case
Fuselage	12738	2430360	1	1 and 2
Wing	5106	610462	1	1 and 2
Tip store	4342	130650	1	1 and 2
Vertical stabiliser	8422	350846	1	2
Horizontal stabiliser	8422	350846	1	2

Table 5.3: Components of the Open Source Fighter cases.



(a) Pressure coefficient at $y/y_{tip} = 0.59$

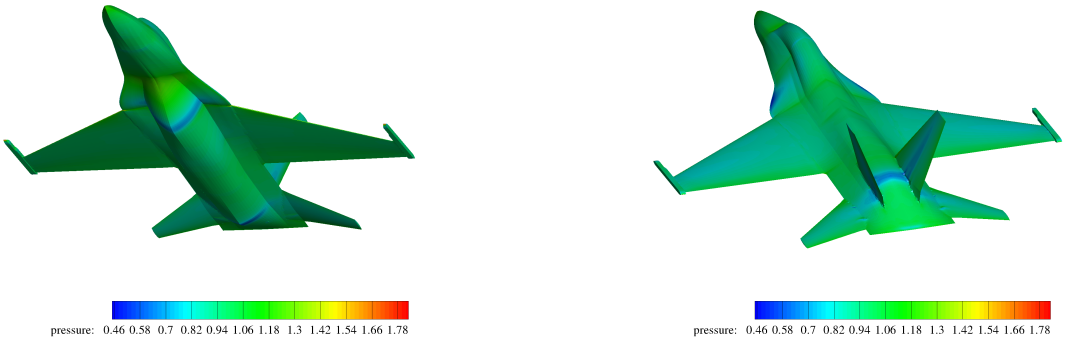
(b) Pressure coefficient at $y/y_{tip} = 0.85$

Figure 5.32: Surface pressure coefficient plots for the OSF Case 1, at M_∞=0.85 and α= 2.12°, compared with results from PMB and experimental data from Ref. [140].

5.6.3 Open Source Fighter

A generic fighter configuration is tested next, using a model that is based on publicly available data for the F-16, and is called the Open Source Fighter (OSF) in this work [141]. The OSF is a model designed to establish a test case which is recognisable as an aircraft, and has similar aerodynamic charateristics to the F-16, without replicating the actual behaviour of the F-16 itself; thus, the model contains several inaccuracies with respect to a realistic geometry.

Two cases are tested in this section: one in which the model is built from three individual components, and one in which it is built from five components; in both cases all of the bodies will fully intersect. Case 1 consists of the fuselage, wing and tip store; and the results obtained can be compared with those from PMB, which is tested on the same configuration using a full mesh. Case 2 consists of the fuselage, wing, tip store, horizontal, and vertical stabilisers; and is to further demonstrate the meshless preprocessor. Figure 5.31(a) shows the resulting surface when all of the components are assembled for Case 2; and the size of each of the input domains for each case is summarised in Table 5.3.



(a) Front view (b) Rear view

Figure 5.33: Pressure contours for OSF Case 2 at M_∞=0.85 and α= 2.12°.

An inviscid, steady state simulation of the resultant geometry was performed at a freestream Mach number of 0.85 and angle of attack 2.12°, using the stencils selected in the preprocessor. The geometry used by both PMB and PML has a wing twist, which was chosen by comparing CFD data with experimental results from Ref. [140]. At these flow conditions the un-twisted wing has a shock wave at the junction of the wing with the tip store: to mitigate these effects, the effective angle of attack was reduced by adding negative twist at the wing tip [142]. The pressure coeffcients at two spanwise locations are presented in Fig. 5.32, compared with results from PMB and experiment. The results at $y/y_{tip} = 0.59$ agree well with the inviscid data, but the slight shocks that are located further along the wing at $y/y_{tip} = 0.85$ are not well predicted. This location is near the tip store, so is at a location where three point distributions overlap; this may be the reason for the loss of accuracy here, due to either the method of coupling the resolving vectors or badly defined parameters in the merit function. Therefore, it is necessary for the method to undergo more research and development; some more reasons for the failure and possible remedies are described in Chapter 7.

The second case has two additional components, namely, the horizontal and vertical stabilisers, as can be seen in the flow results in Fig. 5.33. The fuselage grid contains an xz symmetry plane, Fig. 5.31(b); and as the geometry is made from intersecting the components, the vertical stabiliser is not placed within the symmetry plane. Instead, it is positioned so as to form a twin-tail configuration with the symmetry plane. As the intention of this work is only to demonstrate the capabilities of the meshless method, this difference in the model is not a major concern. This is an interesting case as it shows how the meshless method can be used to perform CFD calculations over aircraft configurations in a much easier way than creating a complete mesh around the entire geometry. The ability to deal with several component domains in this way is also advantageous if one wishes to move components in a moving-body simulation: a prototype store release using this configuration will be performed in Chapter 6; the timings for the preprocessor with the OSF will also be dealt with in that chapter.

Chapter 6

Moving Body Problems

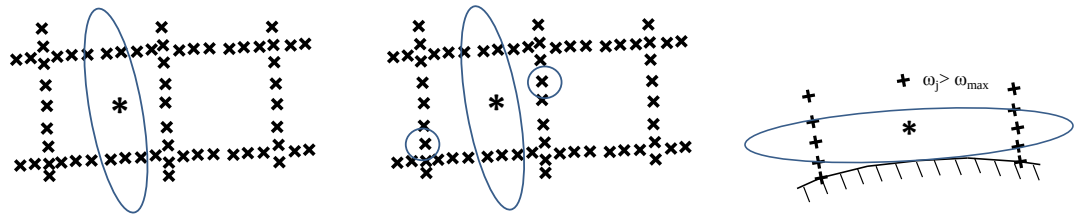
6.1 Stencil selection with moving bodies

The moving of the bodies in time-dependent, unsteady simulations is performed by moving the points of each input grid separately over one another during the computation. When a body moves, the points that share the same grid move rigidly with the body. The extension of the method presented for steady, multibody systems in Chapter 5 to moving-body problems is relatively straightforward. The same method of selecting the stencils is used, but with a small alteration, which exploits the flow solution at the previous time step to help select the stencils for the current time step. One must also consider how to deal with points entering and leaving the domain during the simulation. These developments are considered in the current chapter.

6.1.1 Adaptive stencil selection

The stencil selection method presented in Chapter 5 only uses geometric information regarding the point locations in the selection: there is no a-priori knowledge about the flow solution used. As previously stated, the quality of the stencils is ultimately judged by the accuracy with which the flow can be calculated, and this depends on the solution that the stencil is trying to approximate. For example, stencils can have a more isotropic nature for the solution of the Euler equations; while for the high gradient flows that are associated with the Navier-Stokes equations, anisotropic stencils are required, but in regions of constant flow, such anisotropic stencils are not ideal.

The flow type at a location can only be determined from the flow solution itself; thus, a method that uses both geometric information *and* some flow solution information is attractive for accurate stencil selection, and exploits the flexibility inherent in meshless methods. In this work, the adaptive stencil selection is very simple and differs from the method in Chapter 5 only in that the parameter ψ_{max} is removed, and some flow solution is used instead. For unsteady problems with moving geometries, this flow solution is obtained from the previous real-time step to help construct the stencils at



(a) Anisotropic stencil in a region of constant flow, obtained using only geometric criteria

(b) The circled points can be added to the stencil to improve conditioning in constant flow region

(c) For boundary layer flow the point indicated cannot be added to the stencil because it violates the condition on ω

Figure 6.1: Reconstruction of the stencils in regions with different flow types using the adaptive selection method and new parameter ω_{max}.

the current time-step. For this, a quantity denoted ω_j is defined for every point j in the stencil that has been constructed at the current time step, using the method of Chapter 5 without the constraint of ψ_{max}; the quantity is then determined from the previous flow solution as

$$\omega_j = |\Delta \mathbf{p}|_2^2$$

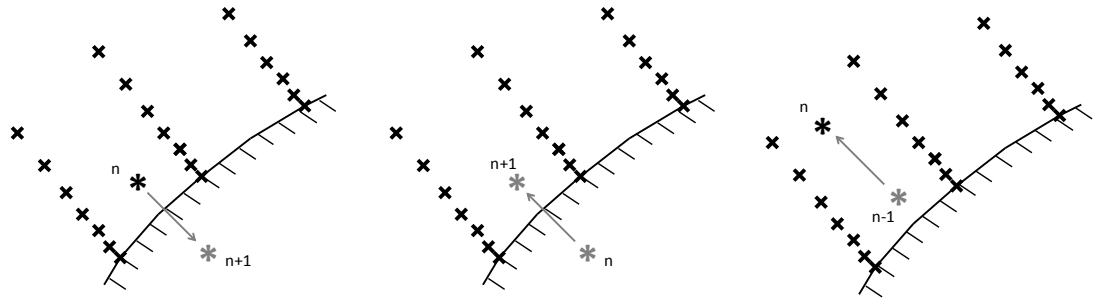
where $\Delta \mathbf{p} = \mathbf{p}_j - \mathbf{p}_i$ is the vector of the difference in primitive variables between the star point i and the neighbouring point j.

Recall from Section 5.3.3 that ψ_{max} was introduced to prevent points being chosen that lie far from the star point in the quadrants that are in the resolving vector direction, Fig. 5.12(c). Though ψ_{max} preserved the quality of stencils in high gradient regions, in areas of constant flow it is unnecessary and can reduce the conditioning of some of the stencils, Fig. 6.1(a). The value of ψ_{max} was determined using geometric information in Chapter 5; now, using the flow data, we can instead define ω_{max}, which is the highest value of ω obtained from the points selected in quadrants A and B in Fig. 5.12(c). Points with the lowest value of ψ from quadrants C and D are then chosen if the value of ω for these points is less than or equal to ω_{max}. This means that for regions of constant flow, where anisotropic stencils may not be ideal, there will always be points in all quadrants, and a maximum of nine points in the stencil in two-dimensions, Fig. 6.1(b); but boundary layer stencils are *still* anisotropic for accurate solution, Fig. 6.1(c). As this method only really affects stencils in regions of the domain where the flow gradients are low, the results are not greatly changed; instead, the major advantages are to the stability of the least squares system, and hence the costs of each solver step.

6.1.2 Points changing type during simulation

The complete, unsteady solver algorithm used is as follows:

1. Construct the stencils for the first time, using only the method in Chapter 5.



(a) An interior point blanked during the computation

(b) A blanked point entering the domain at step n

(c) The point is now active in the flow calculations in the next step

Figure 6.2: Points moving in and out of the computational domain.

2. Perform the steady state CFD calculation.

3. Unsteady solver

   (a) Change the point locations by moving the points of the grid belonging to the body to be moved.

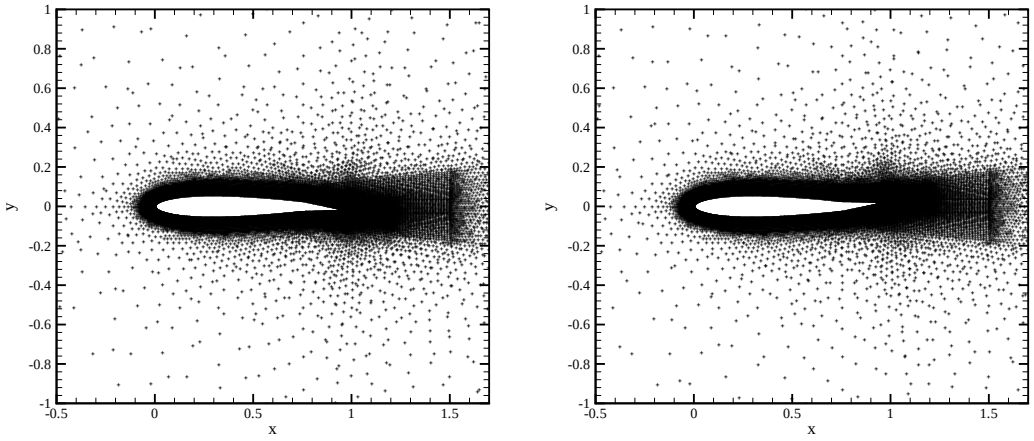
   (b) Recalculate the stencils as described in Chapter 5 using the underlying grid connectivity *and* the obtained flow solution, as in Section 6.1.1.

   (c) Update the flow variables for the points that enter the domain.

   (d) Perform the unsteady, time accurate CFD calculation for this time step.

Step 3 is continued until the simulation finishes. When the unsteady solver loop begins, the full stencil selection stage takes, on average, between 5%–15% of the time taken to update the flow solution (which includes the solver and stencil selection) for each real-time step Δt. This percentage depends on the size of the domains, the order that the points are input, and the number of boundary elements.

There are some additional aspects that need to be considered for moving-body problems, which are contained in Step 3c. If a point that was in the computational domain moves behind a boundary wall for the subsequent time step after the point locations are changed, as shown in Fig. 6.2(a), then the point is now blanked and takes no part in the computation for the new time step. If a point that was blanked on the previous time step now enters the computational domain at step n, Fig. 6.2(b), it still effectively remains blanked, though the flow data at the point is found by interpolation of the flow variables of a neighbouring point; the overlapping input stencil that is used to couple the resolving vectors is used for this. These values are stored so that when the next time step proceeds, as in Fig. 6.2(c), we have the flow variables at step $n-1$, which is needed in Eq. (3.9) for the dual-time stepping method used in the unsteady solver.



(a) Point distribution at $\beta = 5°$

(b) Point distribution at $\beta = -5°$

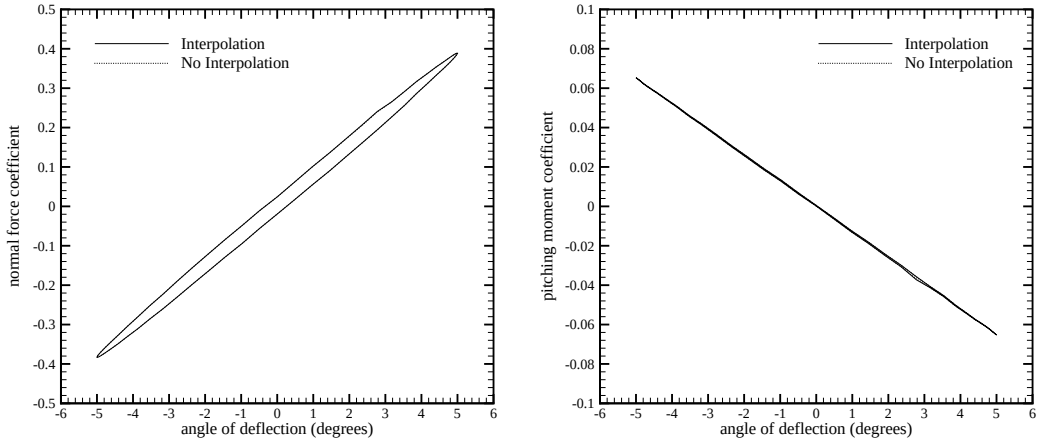
Figure 6.3: Point distributions at maximum control surface deflections.

6.1.3 Point velocities

The moving-body cases are performed in a quasi-unsteady manner in which the boundary movement is fully unsteady, with the velocity of the walls accounted for in the boundary condition from Eqs. (2.30) and (2.31), while the remaining interior points associated with the moving-body are moved only so as to resolve the flow around the body in its new position. Thus, during the simulation the interior points are just locations on which the least squares approximation takes place: these points have no relative velocity to account for this movement in the simulation. The flow history from the previous location of the point is used in Eq. (3.10) for the time-dependence. This means that points may be moving into regions where the flow has different characteristics to the flow at the previous location occupied by the point, and so some sort of interpolation of the flow data may be required: similar to the interpolation required when a point enters the domain from a solid body at the latest step. These issues are not addressed in this work: the focus here is solely on the construction of appropriate stencils; and the moving-body cases that are performed in the next section are mainly to demonstrate the cost of the stencil selection procedure relative to the solver time.

6.2 Moving-body results

To demonstrate the scheme in two-dimensions, a control surface deflection case and prototype store release test case are presented; these cases also test the method of assigning flow variables to points entering the domain, and the adaptive stencil selection scheme, respectively. In three-dimensions, two store release cases at inviscid flow conditions are included to demonstrate the scheme: one consists of a very simple simulation, to give the preprocessor costs relative to the solver time; and the other case is a more complicated case using the OSF of Section 5.6.3.



(a) Normal force coefficient

(b) Pitching moment coefficient

Figure 6.4: Normal force and pitching moment coefficients for unsteady forced control surface deflection case.

6.2.1 Control surface deflection

The first case is an unsteady calculation in two-dimensions of an idealised control surface deflection. Two input domains are used: a main body, consisting of 15552 points, with 392 on the solid wall; and a control surface of 10339 points, with 216 on the solid wall. The main body is a truncated NACA 0012 aerofoil, and the control surface overlaps the main body so that when it exhibits no deflection, the configuration is effectively a full NACA 0012 aerofoil.

The required control surface deflection β is performed by a forced rotation around the rotation centre, located at $x = 0.75$; thus, the configuration is similar to that used in Section 5.4.5. The rotational motion is prescribed by a sinusoidal function

$$\beta(t) = \beta_0 + \beta_a \sin(2kt)$$

with a mean incidence of $\beta_0 = 0°$, pitch amplitude of $\beta_a = 5°$ and reduced frequency of $k = 0.01$. The point distribution of the configuration at the maximum deflections $\beta = \pm 5°$ are shown in Fig. 6.3.

Three motion cycles, each consisting of 64 real-time steps, were simulated to solve the Euler equations at $M_\infty = 0.3$, $\alpha = 0°$. Normal force and pitching moment coefficients for this case are shown in Figs. 6.4(a) and 6.4(b) respectively; these plots show the results both for when the interpolation of flow variables for points entering the domain, as described in Section 6.1.2, is used, and when it is not. The plots show that there is negligible difference in results for this method; however, the use of the interpolated scheme is more cost effective, as, for this case, it takes approximately 78% of the time compared to not interpolating the flow variables for the cycles to converge. This method needs to be tested more, however, to robustly determine its performance.

	Coarse		Medium		Fine	
	time (s)	%	time (s)	%	time (s)	%
Check boundary overlap	0.00001	<0.1	0.00001	<0.1	0.00001	<0.1
Point blanking	0.03	0.30	0.05	0.10	0.24	0.12
Stencil selection	0.81	8.22	4.32	8.36	13.34	6.57
Final boundary check	0.01	0.10	0.07	0.14	0.24	0.12
Total preprocessor	0.85	8.62	4.44	8.59	13.82	6.81
Solver time	9.01	91.38	47.23	91.41	189.14	93.19
Solver time (non-adaptive)	9.04	-	49.67	-	196.48	-

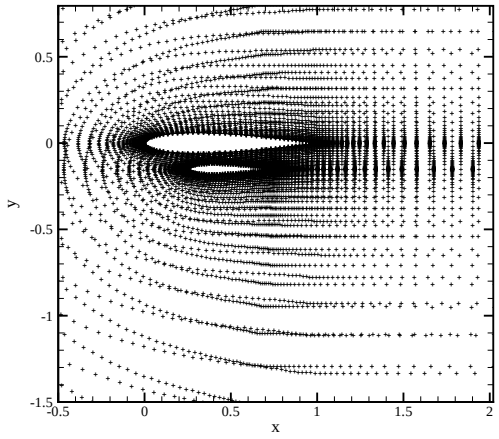
Table 6.1: Timings for each stage of the laminar two-dimensional store case preprocessor in seconds and as a percentage of total time.

6.2.2 Two-dimensional prototype store release

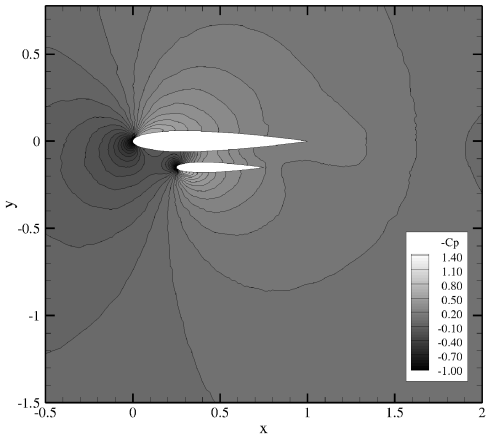
The second case is a prototype store release in two-dimensions, using two NACA 0012 aerofoils, in which the second aerofoil, at half the size, acts as a store located beneath the main aerofoil. The aerofoils are the same as those used for the steady state biplane test case in Section 5.4.1. The store is set in a forced descent from the main aerofoil; and the flow conditions are laminar at a freestream Mach number of 0.5, zero degrees angle of attack and Reynolds number of 5000. The initial location of the store leading edge is at (0.25,-0.15), and the descent velocity is $\dot{y} = -0.1$ in dimensionless units, where the freestream travels one main aerofoil chord in one non-dimensional time unit. There are 200 time steps performed with $\Delta t = 0.05$, so the full time simulation takes place over ten non-dimensionalised time units.

The point distributions and pressure coefficient contours at times 0, 5 and 10 are presented in Fig. 6.5. The plots show how the point distributions change at each real-time step as the points from the grids slide over one another. From the pressure coefficient plot in Fig. 6.5(b), we can see the change in pressure between the aerofoils as the gap between them converges and diverges. Figures 6.5(d) and 6.5(f) show the reduced effect of the upper aerofoil on the pressure distribution of the lower aerofoil as it descends further; and the flow can be seen to be reaching the expected symmetry in the solution for the upper aerofoil as the lower aerofoil moves away.

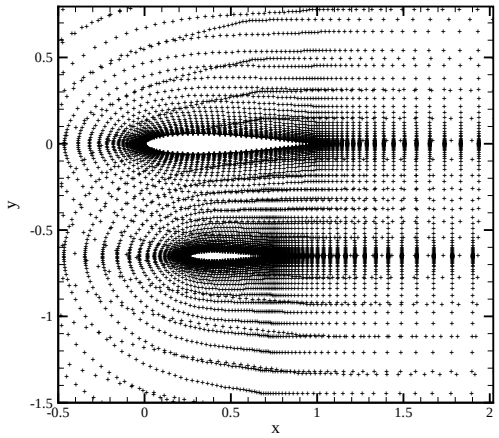
Due to the relative simplicity of this test case, a study of the timings for overlapping grids of increasing size was made. These meshes are the same as those used for the upper aerofoil in Section 5.4.2, and are labelled coarse, medium and fine. The computational domain consists of the overlapping aerofoil grids, and so has double the number of points for each case. The average timings in seconds for each unsteady real-time step are given in Table 6.1. The preprocessor takes a smaller percentage of the time for a time step update as the number of points in the domain increases. For the coarse case it takes 8.62%, the medium case is 8.59% and the fine case is 6.81%.



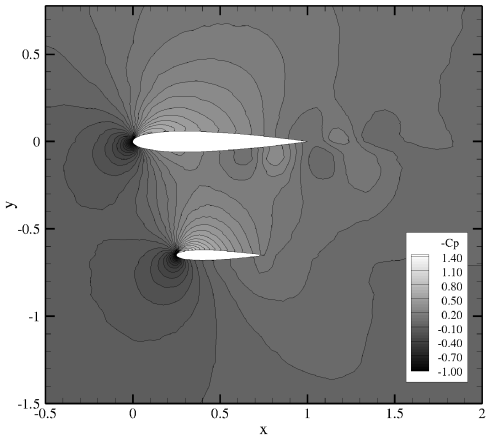
(a) Meshless point distribution at 0.0



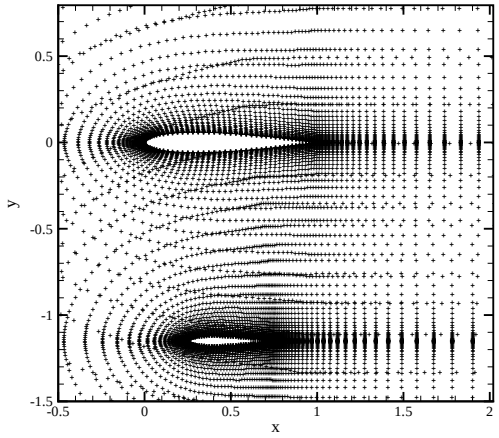
(b) Pressure coefficient at 0.0



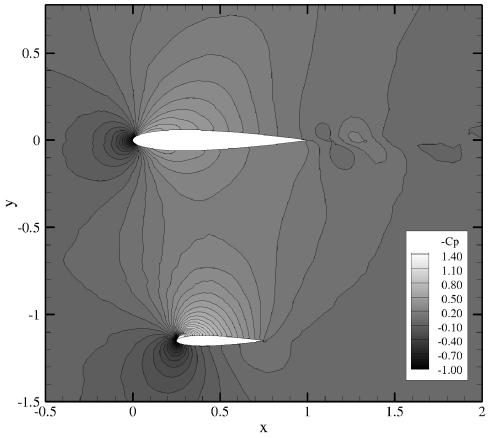
(c) Meshless point distribution at 5.0



(d) Pressure coefficient at 5.0

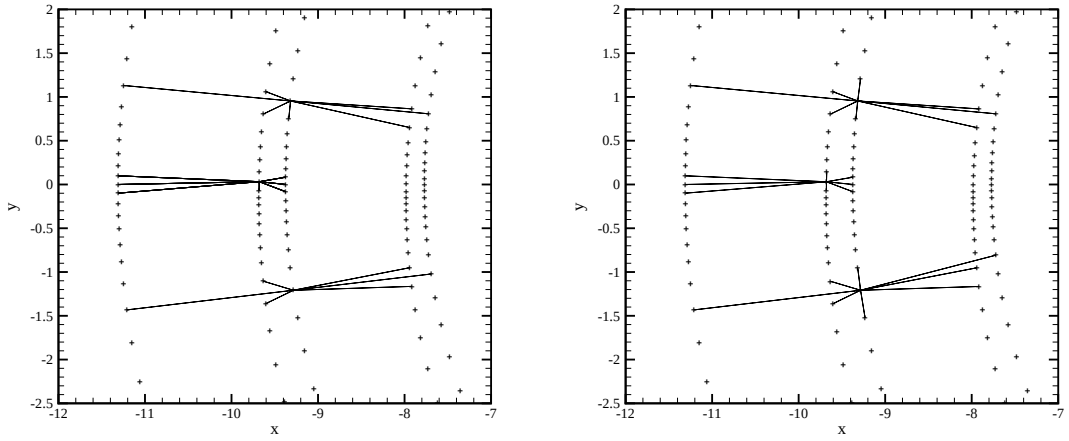


(e) Meshless point distribution at 10.0



(f) Pressure coefficient at 10.0

Figure 6.5: Point distributions and pressure coefficient plots at various unsteady time steps for the two-dimensional store release case, at laminar flow conditions M_∞=0.5, α=0° and Re=5000.



(a) Stencils without adaptive scheme

(b) Stencils with adaptive scheme

Figure 6.6: View of stencils along the outer region of the input domain of the smaller aerofoil, before and after the adaptive scheme has been applied, for the laminar store release case.

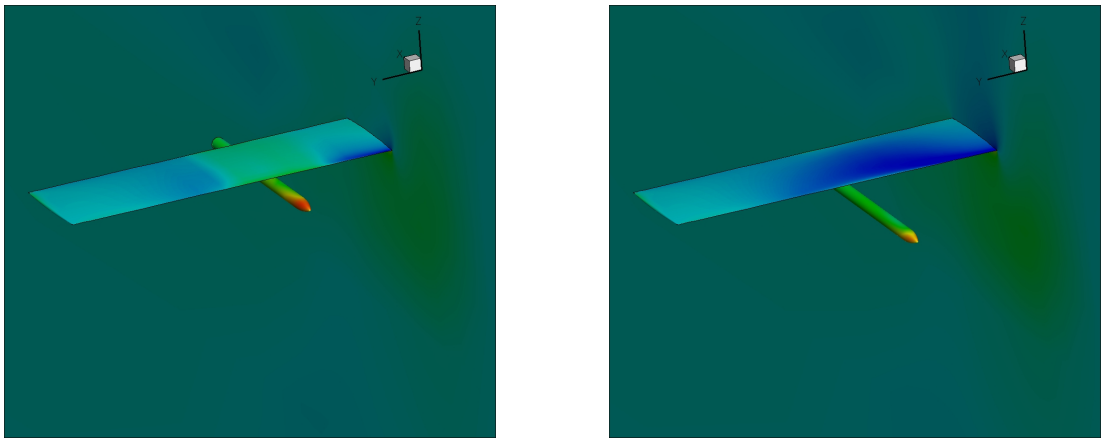
Table 6.1 also includes the solver times for when the adaptive stencil selection scheme is not used. The block structured topology of the input domains means that the majority of the stencils are anisotropic; and as most of these stencils lie relatively far away from the bodies, the results are not changed. The table shows, however, that the adaptive scheme is more cost effective for the solver, particularly for the medium and fine point distributions. This is because, on average, about 80% of the points in these domains are affected, leading to roughly 5% of the solver time being saved; this suggests that, despite the additional points in each stencil, the improved conditioning of the least squares systems offsets this cost. The improvement in the stencil quality is most prevalent for the points along the outer region of the input domain of the smaller aerofoil (where the far field boundary would be located), where the flow is almost constant, but the point distributions do not reflect this, and stencils are more likely to be poorly conditioned or even singular. This can be seen in Fig. 6.6, which shows various stencils in this region, before and after the adaptive scheme has been applied, during the simulation for the medium test case.

6.2.3 Goland wing and store body prototype store release

An unsteady case is performed with two bodies at Mach 0.5 and zero degrees angle of attack to demonstrate the scheme in three-dimensions; these consist of the Goland wing from Section 4.2.1 and a store body (with no fins), consisting of 62623 points with 2122 on the solid wall, which is located just beneath the wing. The initial location of the store is at 15% of the mean chord length below the wing; and from this position it is released, in a forced motion, at $\dot{z} = -0.1$ in dimensionless units, where the freestream travels one mean chord length in one non-dimensional time unit. There

	time (s)	%
Check boundary overlap	0.00001	<0.1
Point blanking	0.96	2.31
Point searching	1.21	2.91
Point sorting	1.91	4.60
Final boundary check	1.23	2.96
Total preprocessor	5.31	12.78
Solver time	36.23	87.22

Table 6.2: Average timings for each stage of the preprocessor in seconds and as a percentage of total time for the Goland wing and store body case.



(a) Pressure coefficient at time step 0.0

(b) Pressure coefficient at time step 5.0

Figure 6.7: Pressure coefficient contours at two time steps for the Goland wing and store body case at inviscid flow conditions M_∞=0.5, α=0°.

are 200 time steps performed with $\Delta t = 0.05$, so the full simulation takes place over ten non-dimensionalised time units; and the pressure coefficient plots at times 0.0 and 5.0 are given in Fig. 6.7.

This case is performed because it is small enough for the flow solver to be able to operate in serial; hence, a reasonable estimate of the cost of the preprocessor relative to the cost of each unsteady step of the flow solver in three-dimensions can be determined. This is given in Table 6.2, which shows the average costs over the entire simulation, and also gives a breakdown of the costs within the preprocessor itself. The method of interpolating flow variables for points that were previously blanked, but eventually enter the domain during the simulation, is used, as is the adaptive stencil selection scheme. For this test case, the preprocessor takes about 13% of the cost of the total time step.

	No. surface points	No. total points	Quantity
Pylon	2690	67250	1
Store body	4762	62632	1
Store fin	2402	21618	4

Table 6.3: Size and quantity of additional components to those in Table 5.3 for the OSF store release case.

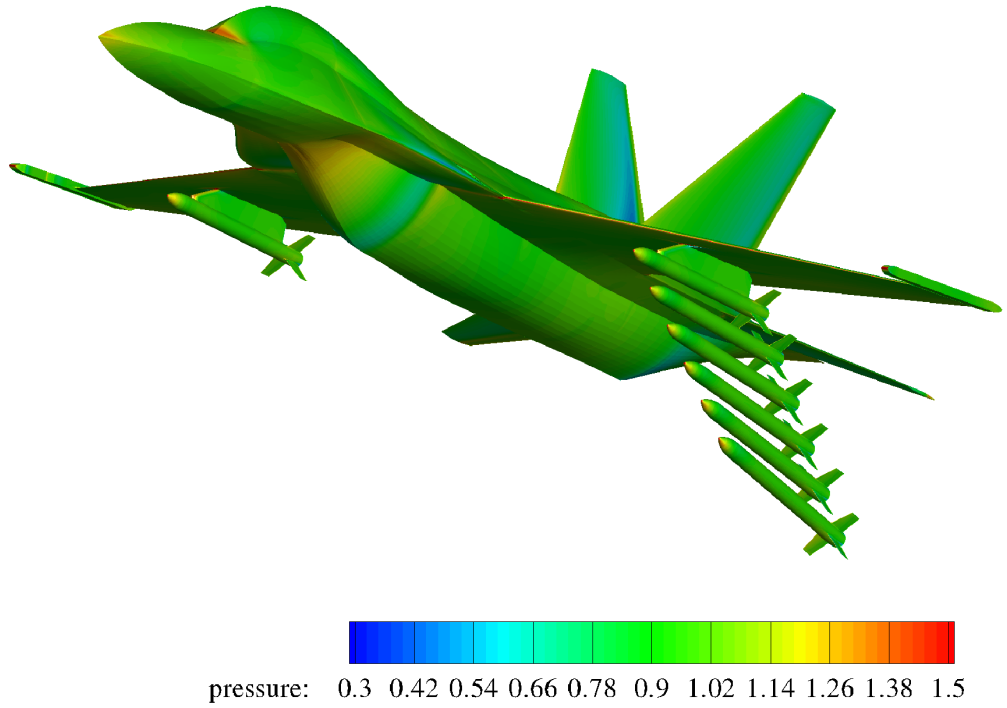


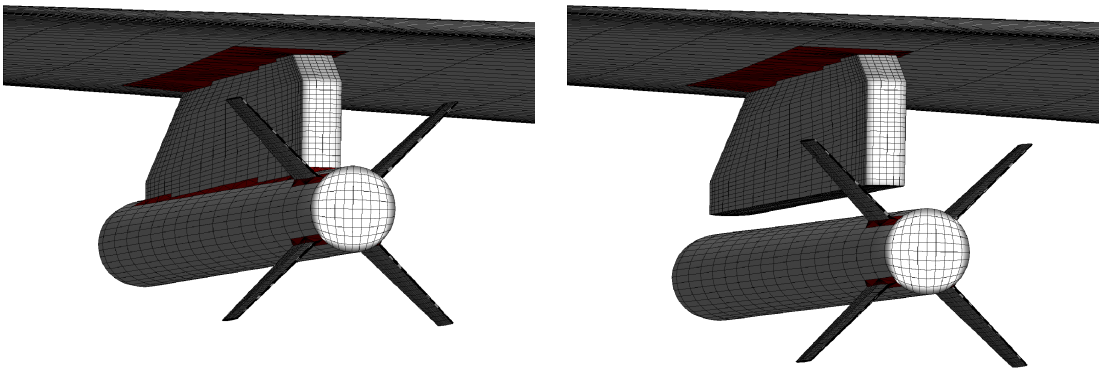
Figure 6.8: Transonic store release from OSF using prescribed motion.

6.2.4 Open Source Fighter store release

The final test case is a store release using the OSF configuration of Section 5.6.3. This initial configuration consisted of a fuselage, wing, tip store, vertical stabiliser and horizontal stabiliser; the size of each of the components was given in Table 5.3. The additional components to those in Section 5.6.3 are a pylon, store body and four store fins; the size of each is given in Table 6.3. The total number of points for this case is over four million, so is too large for the flow solver to operate in serial, and so it must be partitioned at each step as the bodies move. As multiple domain decompositions are required, and the preprocessor does not yet currently work in parallel, the simulation in this section consists only of steady state computations for each position of the store; thus, unlike the other cases in this chapter, this case is not time-dependent and so there is no adaptive stencil selection or flow interpolation involved.

	Case 1		Case 2		Case 3	
	time (s)	%	time (s)	%	time (s)	%
Boundary reallocation	2.48	0.20	5.74	0.30	14.15	0.55
Point blanking	16.19	1.32	25.24	1.30	37.47	1.45
Point searches	273.07	22.28	356.06	18.37	476.45	18.45
Point sorting	433.71	35.38	571.33	29.47	712.22	27.58
Boundary stencils	372.19	30.36	777.92	40.13	1100.13	42.60
Final boundary check	128.26	10.46	202.20	10.43	242.25	9.38
Total preprocessor time	1225.90	-	1938.49	-	2582.67	-

Table 6.4: Timings for each stage of the preprocessor in seconds and as a percentage of total preprocessor time for Case 1 and 2 of Section 5.6.3, and the OSF store release case, labelled Case 3. Note all computations are performed in serial.



(a) Close up of missile attached to pylon

(b) Close up of missile detached from pylon

Figure 6.9: Close up of missile, pylon and wing configuration for the OSF store release test case.

The pylon geometry intersects the wing and stays fixed for each of the computations. The store is a generic missile, which is constructed using a store body and four fins that intersect the store at the rear of the body. For the first step, the store is attached to the pylon, as can be seen in Fig. 6.9(a); this figure shows, in red, all of the elements that were created by the preprocessor to accommodate the intersections between the pylon and wing, the store body and pylon, and the fins and store body. For the subsequent steps the store is detached from the pylon, as can be seen in Fig. 6.9(b). Over the six calculations that this case consists of, the store moves 90% of the mean chord length in the positive x direction and 60% in the negative z direction; it also pitches up 8° around the centre of gravity, which is taken to be 70% along the length of the missile. The pressure contours for six of these calculations at freestream Mach number of 0.85 and angle of attack 2.12° at inviscid flow conditions are given in Fig. 6.8.

The average cost for each operation of the preprocessor for this test case (labelled Case 3) and Cases 1 and 2 of Section 5.6.3 is outlined in Table 6.4 in seconds and the cost as a percentage of the total preprocessor time. The costs are not given relative to the solver time because the solutions are steady state and have been run in parallel; and, as stated, the preprocessor does not currently operate in parallel, hence the high costs given in the table. In agreement with the DLR-F6 test case of Section 5.6.2, the stencil selection takes the most time, though because of the many boundary intersections that occur, the operation constructing stencils for the points that make up the new elements is much more significant. Possible methods of reducing computational times for these schemes, with even greater improvement for moving-body simulations, are given in Chapter 7.

Chapter 7

Conclusions and Future Work

The main aim of this thesis was to investigate the feasibiliy of solving the equations of fluid dynamics over multibody systems using the meshless method. The meshless method differs from conventional finite volume methods in CFD in that it uses a domain consisting only of points on which to solve the equations. Local stencils are required for each point, which are a set of neighbouring points that are used by the meshless flow solver to compute the flux derivatives of the Navier-Stokes equations; this is done using a weighted, polynomial least squares approximation of the flow variables at each point in the stencil. The integration with respect to time of the flow variables is then performed implicitly, using a linearisation of the system and an approximate Jacobian matrix, formed using only the sparsity pattern of the first order matrix. This system is solved inexactly at each iteration using a preconditioned Krylov subspace iterative method.

The computational domain for the presented scheme consists of a point distribution that is formed from the union of separate point distributions that are generated around each body, or component of the body, in the multibody system. In this work, these point distributions are generated from CFD meshes, as the tools for this are readily available; currently, these are user generated meshes using the program ICEM, but plans are that the automatic mesh generator SOLAR, which is used by BAE Systems, will be used instead in the near future. A preprocessor method then redefines boundaries, blanks points that fall within solid bodies and constructs the stencils. Thus, the difficulty of generating meshes in conventional finite volume techniques is replaced by the need to select appropriate stencils for the meshless flow solver; and so the selection stage is very important if the method is to be competitive with other methods, both in terms of accuracy and cost. Because of the anisotropy of the point distributions needed to solve the various forms of the Navier-Stokes equations, classical nearest neighbour algorithms are not used in the stencil selection; instead, the original point connectivities of the input domains are used as a guide to select points that retain the anisotropic nature of the input domains. Resolving vectors are defined for each point to quantify

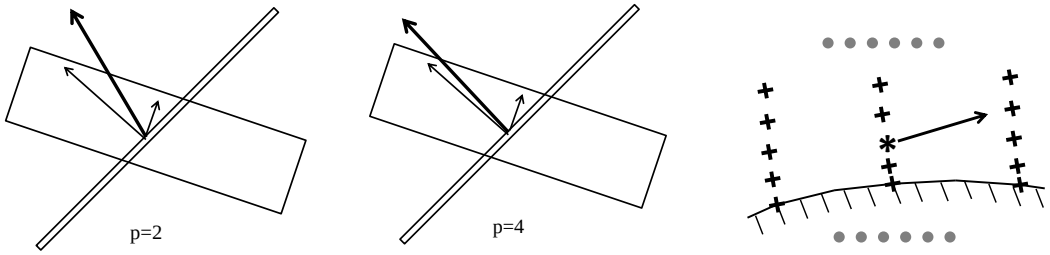
the individual mesh refinement; these are then summed from the various input point distributions that overlap, to obtain a resolving vector for the composite points. The resultant vector defines a new coordinate system; and the neighbouring candidate points, which are found using a combination of various tree searches, are ranked according to the anisotropy and distance of each point from the star, using a merit function that is based on an ellipse in two-dimensions, and an ellipsoid in three-dimensions. The connectivities of the initial grid stencil are therefore used:

- to help define the search regions for efficiency and robustness;
- in the resolving vector definition;
- and to define the parameters for the merit function.

Using this method gives levels of refinement that improve the quality of the stencils in regions where the point distributions are anisotropic, and allows high gradient functions to be well resolved.

For multibody systems in which the bodies move relative to one another in time-dependent computations, the point distributions move with the body that they are associated with, and so the stencil selection is performed after each real-time step when the bodies move. As a result, the selection must be done automatically and efficiently to keep user input and computational overheads as low as possible. The flexibility of the meshless method is particularly advantageous for use with moving-body simulations, as there is no communication between input domains, the motion of the bodies does not need to be known a-priori and large scale motions are possible.

This work was to test the feasibility of using the meshless method for CFD problems; consequently, the results presented are not necessarily real-life aerospace applications, but are instead designed to test the scheme with point distributions that would practically be used to solve the Navier-Stokes equations. The results in two-dimensions include steady state inviscid, laminar and turbulent flows, which are compared with experimental data and other numerical results from the literature. The results of turbulent flow in two-dimensions show the accuracy of the scheme, despite the lack of formal conservation, even for unsteady problems involving moving shocks. Moving-body results are also presented to demonstrate the scheme for such cases; and the costs are presented, in which the times for stencil selection, in both wall-clock time and as a percentage of the flow solver time, can be determined. For this work the three-dimensional cases are restricted to inviscid flows; and though some of the results presented agree well with data from the literature, some of the results, particularly when the bodies intersect, lead to the conclusion that more work is needed for these cases to be fully accurate. The stencils are more sensitive in three-dimensions, and it is more difficult for a developer to determine where the problems lie; consequently, full development of the scheme takes longer than the development of the two-dimensional scheme. Problems



(a) Resultant resolving direction if $p = 2$ in Eq. (5.4)

(b) Resultant resolving direction if $p = 4$ in Eq. (5.4)

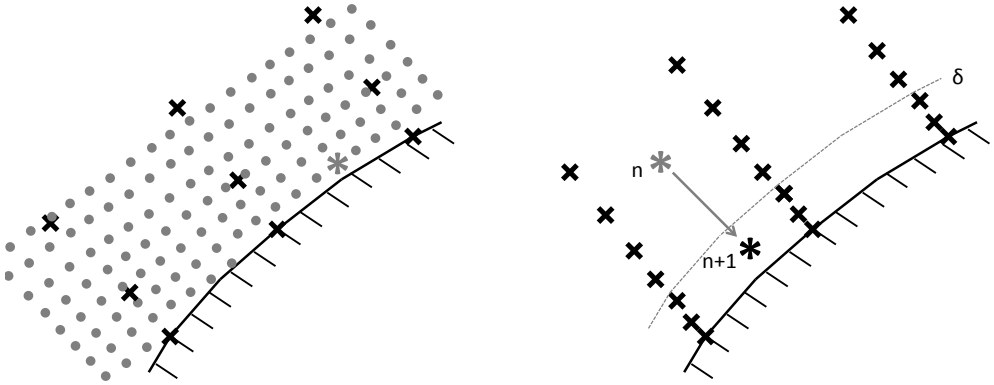
(c) Poor resultant resolving direction

Figure 7.1: Limitations in summing resolving vectors from two overlapping stencils.

could be with the coupling of the resolving vectors, or the choice from the candidate points for the regions where the flow solution is erroneous. Future work could address these issues directly, or could improve on the concepts introducted in this thesis, to develop a more accurate and robust method.

It may be noticed that in the scheme presented in Chapter 5, the method is heuristic in the way that the individual resolving vectors of input domains are coupled to give the final resolving vector. For example, the choice of p in Eq. (5.4) can alter the direction of the resolving vector, as can be seen for the two examples in Figs. 7.1(a) and 7.1(b) (recall that the larger the value of p, the stronger the influence of the finer stencils). Also, a sum of the input vectors is not always desirable, as is the case in Fig. 7.1(c), which is a schematic of a boundary layer region. The points associated with the boundary are marked with crosses, with a layer of stencils of finer resolution, marked by grey dots, overlapping at a right angle; the sum of local resolving vectors would give a poor resultant resolving vector as shown. The method presented in this work is global, but, as different stencils should be used for different flow types, it may be that the method must be localised in some way: both for the solver (choice of weights, order of polynomial reconstruction etc.) and preprocessor (resolving vector influence, parameters of merit function, orientation and number of points chosen etc.); and new testing algorithms could be introduced into the code to improve the stencils and flow solver performance.

The flexibility of meshless methods invites the idea, for example, of using some flow information to do this. This concept was introduced in Chapter 6 for unsteady flows, in which the flow solution at the previous real-time step was used in the selection at the current real-time step. However, the method in Chapter 6 only altered the parameter ψ_{max}, future work could also alter the resolving vectors themselves, or even use a completely different stencil selection method: most likely based on minimising the difference in flow gradients across the points in the stencil. This may be needed for time-dependent calculations, even if not moving-body, due to the velocity changes that can occur. For steady state, multibody systems some partial, initial flow data could be used instead to help recalculate the stencils, and these new stencils are then used for



(a) Problematic point near boundary (b) Point moving into boundary layer

Figure 7.2: Issues regarding point locations.

the calculation to full convergence. Such a partial solution could be achieved using a first order solution using the explicit steps needed to smooth out the initial flow field, as outlined in Chapter 3, to at least guarentee a solution before the flow solution has smoothed out and the Jacobian construction and implicit solver starts. Research into such techniques would remove the need for parameters to be chosen by the user, thus eliminating the problems in Fig. 7.1 and increasing the accuracy.

As outlined in Chapter 5, the scheme in this work was intended to work only on the point locations resulting from the overlap of the various input domains, regardless of whether these are the best point locations. Any future adaptive stencil selection method could also be extended so that the points themselves could be moved, added or removed from the domain. Adaptive schemes to increase accuracy and capture flow phenomena that we may not know would have occured before setting the points are, in some form, already available in the literature, as was discussed in Section 1.3; but future work could also look at altering point locations for when the selection of a well-conditioned stencil is difficult. An example is presented in Fig. 7.2(a), showing two overlapping point distributions near a boundary wall denoted by dots and crosses: the boundary belongs to the input domain containing the crosses. Cases like this occur frequently when point distributions of various resolution overlap, particularly if they belong to point distributions whose boundaries intersect. The highlighted point will have a poor stencil due to this point configuration, as there are not points in all of the sectors surrounding it. When the solver fails for multibody systems it is frequently at such point locations; and for a scheme when the meshless method is to be used globally it *may* be best to remove these points for it to be robust.

To reduce the costs of the preprocessor a parallelisation of the stencil selection is currently under development, but there are also changes that could be made to the code in its present form. The presented scheme is robust in the candidate point selection from the size of the search regions, but it currently means that there are too many candidate

points being found, checked and sorted unnecessarily. This is, by far, the costliest part of the preprocessor, especially for three-dimensional cases when thousands of candidate points can be identified when fine and coarse regions of the point distributions overlap, as was seen in Section 6.2.4. A more efficient procedure, for example, could involve multiple smaller searches, so points will not be checked unnecessarily. The current scheme also means that multiple searches of very similar search regions are taking place for two points that are located very close together. When point distributions are anisotropic this is wasted time, and improvements could be made so that only one search is performed, and the same candidate points are used. For moving-body simulations, the stencil selection is performed globally, but this is not needed if the point domains for a particular stencil are stationary and the flow is relatively constant. Savings in computational time could be made by communicating between the solver and preprocessor if a point stencil needs to be reconstructed. Implementing tests such as these could greatly reduce the cost of the preprocessor, and would be beneficial for the method when it is to be used in moving-body simulations.

Regarding the solver method for moving-body problems, future work must address the issue of how to deal with the change in location of the points accurately. The procedure for points entering the computational domain was outlined in Section 6.1: this needs to be fully tested. In addition, one must also consider if the flow variables must be changed for points that enter a region of the flow that has very different flow properties; an example is given in Fig. 7.2(b) for a point entering a boundary layer. It may or may not be the case that interpolations are required. Furthermore, a 6-DOF module must be implemented to complete the code for use with moving-body problems.

Bibliography

[1] Cenko, A., "Experience in the use of computational aerodynamics to predict store release characteristics," *Progress in Aerospace Sciences*, Vol. 37, 2001, pp. 477–495.

[2] Rogers, R. M., "A comparison between the Nielson and Woodward programs in predicting flow fields and store loads," *Naval Weapons Center TM 2854*, 1976.

[3] Cenko, A. and Tinoco, E. N., "PAN AIR – weapons, carriage and separation," *AFFDL–TR–79–3142*, 1979.

[4] Burg, C. O. E., "A robust unstructured mesh movement strategy using three-dimensional torsional springs," *AIAA Paper 2004-2529*, 2004.

[5] Degand, C. and Farhat, C., "A three-dimensional torsional spring analogy method for unstructured dynamic meshes," *Computers and Structures*, Vol. 80, 2002, pp. 305–316.

[6] Singh, K. P., Newman, J. C., and Basyal, O., "Dynamic unstructured methods for flows past multiple objects in relative motion," *AIAA Journal*, Vol. 33, 1995, pp. 641–649.

[7] Zeng, D. and Ethier, C. R., "A semi-torsional spring analogy model for updating unstructured meshes in 3d moving domains," *Finite Elements in Analysis and Design*, Vol. 41, 2005, pp. 1118–1139.

[8] Johnson, A. A. and Tezduyar, T. E., "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces," *Comput. Methods Appl. Mech. Engrg.*, Vol. 119, 1994, pp. 73–94.

[9] Tezduyar, T. E., Behr, M., Mittal, S., and Johnson, A. A., "Computation of unsteady incompressible flows with the stabilized finite element methods: space-time formulations, iterative strategies and massively parallel implementations," *New Methods in Transient Analysis*, Vol. 143, 1992, pp. 7–24.

[10] Hassan, O., Sørensen, K. A., Morgan, K., and Weatherill, N. P., "A method for time accurate turbulent compressible fluid flow simulation with moving boundary components employing local remeshing," *Int. J. Numer. Meth. Fluids*, Vol. 53, 2007, pp. 1243–1266.

[11] Barakos, G., Vahdati, M., Sayma, A. I., Breard, C., and Imregun, M., "A fully distributed unstructured Navier-Stokes solver for large-scale aeroelasticity computations," *The Aeronautical Journal*, Vol. 105, 2001, pp. 419–426.

[12] Eliasson, P., Wand, D., Meijer, S., and Nordström, J., "Unsteady Euler computations through non-matching and sliding-zone interfaces," *AIAA Paper 1998-0371, 36th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV*, 1998.

[13] Steijl, R. and Barakos, G., "Sliding mesh algorithm for CFD analysis of helicopter rotorfuselage aerodynamics," *International Journal for Numerical Methods in Fluids*, Vol. 58, 2008, pp. 527–549.

[14] Steger, J. L., Dougherty, F. C., and Benek, J. A., "A chimera grid scheme," *Advances in Grid Generation ASME FED*, Vol. 5, 1983, pp. 59–69.

[15] Rogers, S. E., Suhs, N. E., and Dietz, W. E., "Pegasus 5: An automated preprocessor for overset-grid computational fluid dynamics," *AIAA Journal*, Vol. 41, 2003, pp. 1037–1045.

[16] Noack, R. W., "DiRTlib : A Library to Add an Overset Capability to your Flow Solver," *AIAA Paper 2005–5116*, 2005.

[17] Noack, R. W., "SUGGAR : A General Capability for Moving Body Overset Grid Assembly," *AIAA Paper 2005–5117*, 2005.

[18] Wang, Z. J., Parthasarathy, V., and Hariharan, N., "A Fully Automated Chimera Methodology for Multiple Moving Body Problems," *AIAA Paper 98–0217*, 1998.

[19] Brown, D. L., Henshaw, W. D., and Quinlan, D. J., "Overture : Object–Oriented Tools for Overset Grid Applications," *AIAA Paper 99–3130*, 1999.

[20] Meakin, R., "A New Method for Establishing Intergrid Communication Among Systems of Overset Grids," *AIAA Paper 91–1586*, 1991, pp. 662–671.

[21] Belk, D. M. and Maple, R. C., "Automated Assembly of Structured Grids for Moving Body Problems," *AIAA Paper 95–1680*, 1995.

[22] Tian, S., Wu, Y., and Xia, J., "Numerical Simulation of Unsteady Flow Field around Helicopter in Forward Flight Using a Parallel Dynamic Overset Unstructured Grids Method," *Modern Physics Letters B*, Vol. 23, 2009, pp. 325–328.

[23] Pomin, H. and Wagner, S., "Aeroelastic analysis of helicopter rotor blades on deformable chimera grids," *Journal of Aircraft*, Vol. 41, 2004, pp. 577–584.

[24] Meakin, R. L., "Moving body overset grid methods for complete aircraft tiltrotor simulations," *AIAA Paper 1993-3350*, 1993.

[25] Nakamura, S. and Scott, J. N., "Simulation of unsteady flows through stator and rotor blades of a gas turbine using the Chimera method," *5th Annual Thermal and Fluids Analysis Workshop*, 1993, pp. 255–267.

[26] Dougherty, F. C., Benek, J. A., and Steger, J. L., "On application of Chimera grid schemes to store separation," *NASA Report ARC 255*, 1985.

[27] Wang, Z. J. and Parthasarathy, V., "A fully automated Chimera methodology for multiple moving body problems," *International Journal for Numerical Methods in Fluids*, Vol. 33, 2000, pp. 919–938.

[28] Koomullil, R., Cheng, G., Soni, B., Noack, R., and Prewitt, N., "Moving-body simulations using overset framework with rigid body dynamics," *Mathematics and Computers in Simulation*, Vol. 78, 2008, pp. 618–626.

[29] Noack, R. W. and Boger, D., "Improvements to SUGGAR and DiRTlib for overset store separation simulations," *47th AIAA Aerospace Sciences Meeting, AIAA–2009–340, Orlando, FL*, 2009.

[30] Lijewski, L. E. and Suhs, N., "Time accurate Computational Fluid Dynamics Approach to Transonic Store Separation Trajectory Prediction," *Journal of Aircraft*, Vol. 31, 1994, pp. 886–891.

[31] Meakin, R. L., "Computations of the unsteady flow about a generic wing/pylon/finned-store configuration," *AIAA Paper 1992-4568*, 1992.

[32] Atwood, C. A., "Computation of a controlled store separation from a cavity," *Journal of Aircraft*, Vol. 32, 1995, pp. 846–852.

[33] Meakin, R. L., *Composite Overset Structured Grids", Chapter 11, Handbook of Grid Generation*, CRC Press, 1999.

[34] Gingold, R. A. and Monaghan, J. J., "Smoothed Particle Hydrodynamics : theory and application to non–spherical stars," *Mon. Not. R. astr. Soc.*, Vol. 181, 1977, pp. 375–389.

[35] Lucy, B. L., "A Numerical Approach to Testing the Fission Hypothesis," *Astron. J*, Vol. 82, 1977, pp. 1013–1924.

[36] Liu, M. B. and Liu, G. R., *Smoothed Particle Hydrodynamics : A Meshfree Particle Method*, World Scientific Publishing, 2003.

[37] Li, S. and Liu, W. K., "Meshless and particle methods and their applications," *Applied Mechanics Review*, Vol. 55, 2002, pp. 1–34.

[38] Liu, W. K., Jun, S., Li, S., Jonathan, A., and Belytschko, T., "Reproducing kernel particle methods for structural dynamics," *Int. J. Numer. Methods Eng.*, Vol. 38, 1995, pp. 1655–1679.

[39] Nayroles, B., Touzot, G., and Villon, P., "Generalizing the finite element method : diffuse approximation and diffuse elements," *Comput. Mech.*, Vol. 10, 1992, pp. 307–318.

[40] Belytschko, T., Gu, L., and Lu, Y. Y., "Fracture and crack growth by element-free Galerkin methods," *Model. Simul. Mater. Sci. Engrg.*, Vol. 2, 1994, pp. 519–534.

[41] Belytschko, T., Lu, Y. Y., and Gu, L., "Crack propagation by element-free Galerkin methods," *Engineering Fracture Mechanics*, Vol. 51, 1995, pp. 295–315.

[42] Belytschko, T. and Tabbara, M., "Dynamic fracture using element-free Galerkin methods," *International Journal for Numerical Methods in Engineering*, Vol. 39, 1996, pp. 923–938.

[43] Lu, Y. Y., Belytschko, T., and Tabbara, M., "Element-free Galerkin method for wave propagation and dynamic fracture," *Computer Methods in Applied Mechanics and Engineering*, Vol. 126, 1995, pp. 131–153.

[44] Zhang, L., Ouyang, J., and Zhang, X., "On a two-level element-free Galerkin method for incompressible fluid flow," *Applied Numerical Mathematics*, Vol. 59, 2009, pp. 1894–1904.

[45] Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., and Krysl, P., "Meshless methods: an overview and recent developments," *Comput Methods Appl Mech Eng*, Vol. 139, 1996, pp. 347.

[46] Liszka, T. J., Duarte, C. A. M., and Tworzydlo, W. W., "hp-Meshless cloud method," *Comput Methods Appl Mech Eng*, Vol. 139, 1996, pp. 263–288.

[47] Atluri, S. N. and Zhu, T., "New Meshless Local Petrov-Galerkin (MLPG) approach in computational mechanics," *Comput. Mech.*, Vol. 22, 1998, pp. 117–127.

[48] Lin, H. and Atluri, S. N., "The Meshless Local Petrov-Galerkin (MLPG) method for solving incompressible Navier-Stokes equations," *CMES*, Vol. 2, 2001, pp. 117–142.

[49] Mavriplis, D. J., "Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes," *AIAA Paper 2003-3986*, 2003.

[50] Batina, J. T., "A gridless Euler/NavierStokes solution algorithm for complex two dimensional applications," Tech. Rep. 107631, NASA Technical Memorandum, 1992.

[51] Batina, J. T., "A gridless Euler/Navier-Stokes solution algorithm for complex aircraft applications," Tech. Rep. 107727, NASA Technical Memorandum, 1993.

[52] Oñate, E., Idelson, S., Zienkiewicz, O. C., and Taylor, R. L., "A finite point method in computational mechanics. Applications to convective transport and fluid flow," *Int. J. Numer. Methods Engrg*, Vol. 39, 1996, pp. 3839–3866.

[53] Oñate, E., Idelson, S., Zienkiewicz, O. C., Taylor, R. L., and Sacco, C., "A stabilized finite point method for analysis of fluid mechanics problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 139, 1996, pp. 315–346.

[54] Boroomand, B., Tabatabaei, A. A., and Oñate, E., "Simple modifications for stabilization of the finite point method," *Int J Numer Methods Eng.*, Vol. 63, 2005, pp. 351–379.

[55] Ortega, E., Oñate, E., and Idelsohn, S., "An improved finite point method for tridimensional potential flows," *Computational Mechanics*, Vol. 40, 2007, pp. 949–963.

[56] Ghosh, A. K., *Robust Least Squares Kinetic Method for Inviscid Compressible Flows*, Ph.D. thesis, PhD Thesis, Department of Aerospace Engineering, Indian Institute of Science, Bangalore, 1996.

[57] Ghosh, A. K. and Deshpande, S. M., "Least Squares Kinetic Upwind Method for Inviscid Compressible Flows," *AIAA Paper 95–1735*, 1995.

[58] Praveen, C., *Development and Application of Kinetic Meshless Methods for Euler Equations*, Ph.D. thesis, PhD Thesis, Department of Aerospace Engineering, Indian Institute of Science, Bangalore, 2004.

[59] Katz, A. and Jameson, A., "Edge-based Meshless Methods for Compressible Flow Simulations," *AIAA Paper 2008-699, 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV*, 2008.

[60] Katz, A., Jameson, A., and Wissink, A., "Multicloud: Multigrid convergence with a meshless operator," *J. Comput. Phys.*, Vol. 228, 2009, pp. 5237–5250.

[61] Katz, A. and Jameson, A., "A Comparison of Various Meshless Schemes Within a Unified Algorithm," *AIAA Paper 2009-596, 47th AIAA Aerospace Sciences Meeting, Orlando, Florida*, 2009.

[62] Liu, G. R., *Meshfree Methods - Moving beyond the Finite Element Method*, CRC Press, 1st ed., 2002.

[63] Fries, T. P. and Matthies, H. G., "Classification and overview of meshfree methods," Tech. Rep. Informatikbericht 2003-3, Technical University Braunschweig, Germany, 2004.

[64] Duarte, C. A. M. and Oden, J. T., "An hp adaptive method using clouds," *Comput. Methods Appl. Mech. Engrg.*, Vol. 139, 1996, pp. 237262.

[65] Ortega, E., Oñate, E., and Idelsohn, S., "A finite point method for adaptive three-dimensional compressible flow calculations," *Int. J. Numer. Meth. Engng.*, Vol. 60, 2009, pp. 937–971.

[66] Perazzo, F., Löhner, R., and Pérez-Pozo, L., "Adaptive Methodology for Meshless Finite Point Method," *Adv. Eng. Softw.*, Vol. 39, 156–166, pp. 2008.

[67] Afshar, M. and Firoozjaee, A., "Adaptive Simulation of Two Dimensional Hyperbolic Problems by Collocated Discrete Least Squares Meshless Method," *Computers & Fluids*, Vol. 39, 2010, pp. 2030–2039.

[68] Ma, Z., Chen, H., and Zhou, C., "A study of point moving adaptivity in gridless method," *Comput. Methods Appl. Mech. Engrg.*, Vol. 197, 2008, pp. 1926–1937.

[69] Kirshman, D. J. and Liu, F., "A gridless boundary condition method for the solution of the Euler equations on embedded Cartesian meshes with multigrid," *J. Comput. Phys.*, Vol. 201, 2004, pp. 119–147.

[70] Koh, E. P. C., Tsai, H. M., and Liu, F., "Euler solution using Cartesian grid with a gridless least squares boundary treatment," *AIAA Journal*, Vol. 43, 2005, pp. 246–255.

[71] Sridar, D. and Balakrishnan, N., "An upwind finite difference scheme for meshless solvers," *J. Comput. Phys.*, Vol. 189, 2003, pp. 1–29.

[72] Ninawe, A., Munikrishna, N., and Balakrishnan, N., "Viscous Flow Computations Using a Meshless Solver, LSFD-U," *Proceedings of 3rd International Conference on Computational Fluid Dynamics, Toronto, Canada*, 2004.

[73] Munikrishna, N. and Balakrishnan, N., "Turbulent flow computations on a hybrid cartesian point distribution using meshless solver LSFD-U," *Computers and Fluids*, Vol. 40, 2011, pp. 118–138.

[74] Wang, X. Y., Yeo, K. S., Chew, C. S., and Shu, C., "A SVD-GFD scheme for computing 3D incompressible viscous flows," *Computers & Fluids*, Vol. 37, 2008, pp. 733–746.

[75] Yu, P., Yeo, K. S., Shyam Sundar, D., and Ang, S. J., "A three-dimensional hybrid meshfree-Cartesian scheme for fluid-body interaction," *Int. J. Numer. Meth. Engng*, Vol. 88, 2011, pp. 385–408.

[76] Tang, L. and Tang, J., "Hybrid Cartesian Grid/Gridless Algorithm for Store Separation Prediction," *AIAA Paper 2010-0508, 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010.

[77] Wang, H., Chen, H., and Periaux, J., "A study of gridless method with dynamic clouds of points for solving unsteady CFD problems in aerodynamics," *Int. J. Numer. Meth. Fluids*, Vol. 64, 2010, pp. 98–118.

[78] Idelsohn, S. and Oñate, E., "To mesh or not to mesh. That is the question," *Comput Methods Appl Mech Eng*, Vol. 195, 2006, pp. 4681–4696.

[79] Löhner, R., Sacco, C., Oñate, E., and Idelsohn, S., "A finite point method for compressible flow," *Int. J. Numer. Meth. Engng.*, Vol. 53, 2002, pp. 1765–1779.

[80] Kamruzzaman, M., Sonar, T., Lutz, T., and Krämer, E., "A New Meshless Collocation Method for Partial Differential Equations," *Comm. in Num. Meth. in Eng.*, Vol. 24, 2008, pp. 1617–1639.

[81] Seibold, B., "Minimal positive stencils in meshfree finite difference methods for the Poission equation," *Comput. Methods Appl. Mech. Engrg.*, Vol. 198, 2008, pp. 592–601.

[82] Yin, L., Shen, L., and Lv, G., "Five-point positive meshless scheme for hyperbolic conservation laws," *Int. J. Numer. Meth. Biomed. Engng.*, Vol. 27, 2011, pp. 619–631.

[83] Anandhanarayanan, K., "Development of Three-dimensional Grid-free Solver and its Applications to Multi-body Aerospace Vehicles," *Defence Science Journal*, Vol. 60, 2010, pp. 653–662.

[84] Chiu, E. and Jameson, A., "An Edge-Averaged Semi-meshless Framework for Numerical Solution of Conservation Laws," *AIAA Paper 2010-1269, 48th AIAA Aerospace Sciences Meeting, Orlando, Florida*, 2010.

[85] Seibold, B., *M-Matrices in Meshless Finite Difference Methods*, Ph.D. thesis, PhD Thesis, Department of Mathematics, University of Kaiserslautern, Germany, 2006.

[86] Kennett, D. J., Angulo, J., Timme, S., and Badcock, K., "Semi-Meshless Stencil Selection for Anisotropic Point Distributions," *International Journal of Computational Fluid Dynamics*, Vol. 26, 2012, pp. 463–487.

[87] Kennett, D. J., Angulo, J., Timme, S., and Badcock, K., "An Implicit Meshless Method for Application in Computational Fluid Dynamics," *International Journal for Numerical Methods in Fluids*, Vol. 71, 2012, pp. 1007–1028.

[88] Kennett, D. J., Angulo, J., Timme, S., and Badcock, K., "Semi-Meshless Stencil Selection on Three-Dimensional Anisotropic Point Distributions with Parallel Implementation," *AIAA Paper 2013–0867, Presented at the 51st Aerospace Sciences Meeting, Grapevine, Texas, Jan 2013*.

[89] Kennett, D. J., Angulo, J., Timme, S., and Badcock, K., "An Implicit Semi-Meshless Scheme with Stencil Selection for Anisotropic Point Distributions," *AIAA Paper 2011–3234, Presented at the 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 2011*.

[90] McCracken, A., Akram, U., Da Ronch, A., and Badcock, K., "Requirements for Computer Generated Aerodynamic Models for Aircraft Stability and Control Analysis," *5th Symposium on Integrating CFD and Experiments in Aerodynamics, Tokyo, Japan*, 2012.

[91] Timme, S., Badcock, K., Wu, M., and Spence, A., "Lyapunov Inverse Iteration for Stability Analysis using Computational Fluid Dynamics," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Honolulu, Hawaii*, 2012.

[92] Spalart, P. R., "Strategies for turbulence modelling and simulations," *Int. J. Heat Fluid Flow*, Vol. 21, 2000, pp. 252–263.

[93] Wilcox, D. C., *Turbulence modeling for CFD*, DCW Industries, Inc., La Cañada, CA, 3rd ed., 2006.

[94] Spalart, P. R. and Allmaras, S. R., "A one–equation turbulence model for aerodynamic flows," *AIAA Paper 92–0439, 30th Aerospace Sciences Meeting, Reno, NV*, 1992.

[95] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 2nd ed., 1992.

[96] Toro, E. F., "Riemann solver and numerical methods for fluid dynamics," 1999.

[97] Osher, S. and Solomon, F., "Upwind Difference Schemes for Hyperbolic Systems of Conservation Laws," *Mathematics of Computation*, Vol. 38, 1982, pp. 339–374.

[98] Roe, P. L., "Approximate Riemann solvers, parameter vectors and difference schemes," *J. Comput. Phys.*, Vol. 43, 1981, pp. 357–372.

[99] Mavripilis, D. J., "Unstructured Mesh Discretisations and Solvers for Computational Aerodynamics," *AIAA Paper 2007-3955, 18th AIAA Computational Fluid Dynamics Conference, Miami, Florida*, 2007.

[100] Barth, T. J. and Jespersen, D. C., "The design and application of upwind schemes on unstructured meshes," *AIAA Paper 89–0366, 27th Aerospace Sciences Meeting, Reno, NV*, 1989.

[101] Venkatakrishnan, V., "On the accuracy of limiters and convergence to steady state solutions," *AIAA paper 93-0880, 31st Aerospace Sciences Meeting Exhibit*, 1993.

[102] Edwards, J. R. and Chandra, S., "Comparison of eddy viscosity-transport turbulence models for three-dimensional, shock-separated flowfields," *AIAA Journal*, Vol. 34, 1996, pp. 756–763.

[103] Katz, A., *Meshless Methods for Computational Fluid Dynamics*, Ph.D. thesis, Stanford University, 2009.

[104] Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," *AIAA Paper 91-1596, AIAA 10th Computational Fluid Dynamics Conference, Honolulu*, 1991.

[105] Mavriplis, D. J., "On Convergence Acceleration Techniques for Unstructured Meshes," Tech. Rep. NASA/CR–1998–208732, NASA Langley Research Center, Hampton, VA, 1998.

[106] Eisenstat, S., Elman, H., and Schultz, M., "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations," *SIAM Journal of Numerical Analysis*, Vol. 20, 1983, pp. 345–357.

[107] Badcock, K. J., Xu, X., Dubuc, L., and Richards, B. E., "Preconditioners for high speed flows in aerospace engineering," *Numerical Methods for Fluid Dynamics*, Vol. 5, 1996, pp. 287–294.

[108] Axelsson, O., *Iterative Solution Methods*, Cambridge University Press, 1994.

[109] Karypis, G. and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol. 20, 1998, pp. 359– 392.

[110] Badcock, K. J., Richards, B. E., and Woodgate, M. A., "Elements of computational fluid dynamics on block structured grids using implicit solvers," *Progress in Aerospace Sciences*, Vol. 36, 2000, pp. 351–392.

[111] Shukla, R. K., Tatineni, M., and Zhong, X., "Very high-order compact finite difference schemes on non-uniform grids for incompressible NavierStokes equations," *J. Comput. Phys.*, Vol. 224, 2007, pp. 1064–1094.

[112] Hashemi, M. Y. and Jahangirian, A., "Implicit Fully Meshless Method for Compressible Viscous Flow Calculations," *J. Comput. Appl. Math.*, Vol. 235, 2011, pp. 4687–4700.

[113] Roshko, A., "On the Development of Turbulent Wakes from Vortex Streets," Tech. Rep. Naca Report-1191, 1954.

[114] Cook, P. H., McDonald, M. A., and Firmin, M. C. P., "Aerofoil RAE 2822 – Pressure distributions, and boundary layer and wake measurements," Tech. Rep. AGARD AR 138, 1979.

[115] Landon, R. H., "NACA 0012. Oscillatory and transient pitching," Tech. Rep. AGARD–R–702, 1982.

[116] Badcock, K. J. and Gaitonde, A. L., "An unfactored implicit moving mesh method for two–dimensional unsteady N–S equations," *Int. J. Numer. Meth. Fluids*, Vol. 23, 1996, pp. 607–631.

[117] Davis, S. S., "Compendium of Unsteady Aerodynamic Measurements," Tech. Rep. AGARD–R–702, 1982.

[118] Davis, S. S. and Malcolm, G. N., "Transonic Shock-Wave/Boundary-Layer Interactions on an Oscillating Airfoil," *AIAA Journal*, Vol. 18, 1980, pp. 1306–1312.

[119] Pechloff, A. and Laschka, B., "Small Disturbance Navier-Stokes Method: Efficient Tool for Predicting Unsteady Air Loads," *Journal of Aircraft*.

[120] Goland, M., "The flutter of a uniform cantilever wing," *Journal of Applied Mechanics*, Vol. 12, 1945.

[121] Allwright, S., "MDO process and specification for the primary sensitivity study," Tech. Rep. Technical Report D.2.4.S., MDO/SPEC/BAE/SA960430, 1996.

[122] Allwright, S., "Reference aircraft performance and primary sensitivities," Tech. Rep. Technical Report D.3.12.R., MDO/TR/BAE/SA970530/1, 1997.

[123] de C. Henshaw, M. J., Badcock, K. J., Vio, G. A., Allen, C. B., Chamerlain, J., Kaynes, I., Dimitriadis, G., Cooper, J. E., Rampurawala, M. A. W. A. M., Jones, D., Fenwick, C., Gaitonde, A. L., Taylor, N. V., Amor, D. S., Eccles, T. A., and Denley, C. J., "Non-linear aeroelastic prediction for aircraft applications," *Progress in Aerospace Sciences*, Vol. 43, 2007, pp. 65–137.

[124] Schmitt, V. and Charpin, F., "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers," Tech. Rep. AGARD, TR AR–138, 1979.

[125] Ho-Le, K., "Finite element mesh generation methods: a review and classification," *Computer-Aided Design*, Vol. 20, 1988, pp. 27–38.

[126] van Phai, N., "Automatic mesh generation with tetrahedron elements," *International Journal for Numerical Methods in Engineering*, Vol. 18, 1982, pp. 273–289.

[127] Shephard, M. S. and Georges, M. K., "Automatic three-dimensional mesh generation by the finite octree technique," *International Journal for Numerical Methods in Engineering*, Vol. 32, 1991, pp. 709–749.

[128] Möller, P. and Hansbo, P., "On advancing front mesh generation in three dimensions," *International Journal for Numerical Methods in Engineering*, Vol. 38, 1995, pp. 3551–3569.

[129] Samet, H., "The quadtree and related hierarchical data structures," *Computing Surveys*, Vol. 16, 1984, pp. 187–260.

[130] Bonet, J. and Peraire, J., "An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems," *International Journal for Numerical Methods in Engineering*, Vol. 31, 1991, pp. 1–17.

[131] Aftosmis, M. J., Berger, M. J., and Melton, J. E., "Robust and efficient Cartesian mesh generation for component-based geometry," *AIAA Journal*, Vol. 36, 1997, pp. 952–960.

[132] Aftosmis, M. J., "Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries," *VKI Lecture Series, 1997-02, 28th computational fluid dynamics*, 1997.

[133] O'Rourke, J., *Computational Geometry in C*, Cambridge University Press, 2nd ed., 1998.

[134] Liao, W., Cai, J., and Tsai, H. M., "A multigrid overset grid flow solver with implicit hole cutting method," *Comput. Methods Appl. Mech. Engrg,*, Vol. 196, 2007, pp. 1701–1715.

[135] Jawahar, P. and Kamath, H., "A High-Resolution Procedure for Euler and NavierStokes Computations on Unstructured Grids," Tech. rep., 2000.

[136] Möller, T., "A Fast Triangle-Triangle Intersection Test," *Journal of Graphics Tools*, Vol. 2, 1997, pp. 25–30.

[137] Lain, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O., E.Rakowitz, M., Tinoco, E. N., and Godard, J. L., "Data Summary from Second AIAA Computational Fluid Dynamics Drag Prediction Workshop," *Journal of Aircraft*, Vol. 42, 2005, pp. 1165–1178.

[138] Vassberg, J. C., Tinoco, E. N., Mani, M., Brodersen, O., Eisfeld, B., Wahls, R. A., Morrison, J. H., Zickuhr, T., Lain, K. R., and Mavripilis, D. J.

[139] Martin, M., Andrés, E., Widhalm, M., Bitrián, P., and Lozano, C., "CAD-based Aerodynamic Shape Design Optimization with the DLR Tau Code," *Paper ICAS 2010-2.6.1 27th Congress of International Council of the Aeronautical Sciences,Nice, France, 2010*.

[140] Denegri, C. and Dubben, J., "F-16 Limit Cycle Oscillation Analysis Using Transonic Small-disturbance Theory," *AIAA Paper 2005–2296, 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, AIAA, Austin, Texas*, 2005.

[141] "Open Source Fighter," geometry available at http://cfd4aircraft.com.

[142] Marques, S., Kodaparast, H., Badcock, K., and Mottershead, J., "CFD Based Aeroelastic Stability Predictions Under the Influence of Structural Variability," *50th Structural Dynamics and Materials Conference, Palm Springs, California*, 2009.

Appendix A

Navier-Stokes equations in vector form

In a three-dimensional Cartesian coordinate system, the non-dimensional form of the equations, without body forces or external heat addition, may be written as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial (\mathbf{f}^i - \mathbf{f}^\nu)}{\partial x} + \frac{\partial (\mathbf{g}^i - \mathbf{g}^\nu)}{\partial y} + \frac{\partial (\mathbf{h}^i - \mathbf{h}^\nu)}{\partial z} = 0$$

The vector $\mathbf{w}$ is the vector of conserved flow variables and is written as

$$\mathbf{w} = (\rho, \rho u, \rho v, \rho w, \rho E)^T$$

The flux vectors consist of inviscid (^{i}) and viscous ($^\nu$) diffusive parts. The inviscid flux vectors, $\mathbf{f}^i$, $\mathbf{g}^i$ and $\mathbf{h}^i$, are given by

$$\mathbf{f}^i = \left(\rho u, \rho u^2 + p, \rho uv, \rho uw, \rho uH\right)^T$$

$$\mathbf{g}^i = \left(\rho v, \rho uv, \rho v^2 + p, \rho uw, \rho vH\right)^T$$

$$\mathbf{h}^i = \left(\rho w, \rho uw, \rho vw, \rho w^2 + p, \rho wH\right)^T$$

while the viscous flux vectors, $\mathbf{f}^\nu$, $\mathbf{g}^\nu$ and $\mathbf{h}^\nu$, contain terms for viscous and heat-conduction terms, and can be represented by

$$\mathbf{f}^\nu = \frac{1}{Re} \left(0, \tau_{xx}, \tau_{xy}, \tau_{xz}, u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + q_x\right)^T$$

$$\mathbf{g}^\nu = \frac{1}{Re} \left(0, \tau_{xy}, \tau_{yy}, \tau_{yz}, u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + q_y\right)^T$$

$$\mathbf{h}^\nu = \frac{1}{Re} \left(0, \tau_{xz}, \tau_{yz}, \tau_{zz}, u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + q_z\right)^T$$

where the components of the heat flux vector are written as

$$q_x = -\frac{\mu}{(\gamma-1)M_\infty^2 RePr}\frac{\partial T}{\partial x}$$

$$q_y = -\frac{\mu}{(\gamma-1)M_\infty^2 RePr}\frac{\partial T}{\partial y}$$

$$q_z = -\frac{\mu}{(\gamma-1)M_\infty^2 RePr}\frac{\partial T}{\partial z}$$

and the components of the viscous stress tensor are written as

$$\tau_{xx} = \frac{2\mu}{3Re}\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}\right) \qquad \tau_{xy} = \frac{\mu}{Re}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = \tau_{yx}$$

$$\tau_{yy} = \frac{2\mu}{3Re}\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x} - \frac{\partial w}{\partial z}\right) \qquad \tau_{xz} = \frac{\mu}{Re}\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) = \tau_{zx}$$

$$\tau_{zz} = \frac{2\mu}{3Re}\left(2\frac{\partial w}{\partial z} - \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right) \qquad \tau_{yz} = \frac{\mu}{Re}\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) = \tau_{zy}$$

where M_∞ is the freestream (∞ means freestream) Mach number, Re the Reynolds number and Pr the Prandtl number:

$$M_\infty = \frac{u_\infty}{\sqrt{\gamma R T_\infty}} \qquad Re = \frac{\rho_\infty u_\infty L}{\mu_\infty} \qquad Pr = \frac{\mu C_p}{k}$$

The various flow quantities are related to each other by the perfect gas relations

$$\begin{aligned} H &= E + \frac{p}{\rho} \\ E &= e + \frac{1}{2}(u^2 + v^2) \\ p &= (\gamma - 1)\rho e \\ \frac{p}{\rho} &= \frac{T}{\gamma {M_\infty}^2} \end{aligned}$$

All quantities have been non-dimensionalised as follows

$$x* = \frac{x}{L} \qquad y* = \frac{y}{L} \qquad z* = \frac{z}{L} \qquad t* = \frac{t}{L/u_\infty}$$

$$u* = \frac{u}{u_\infty} \qquad v* = \frac{v}{u_\infty} \qquad w* = \frac{w}{u_\infty} \qquad \mu* = \frac{\mu}{\mu_\infty}$$

$$\rho* = \frac{\rho}{\rho_\infty} \qquad p* = \frac{p}{\rho_\infty u_\infty^2} \qquad T* = \frac{T}{T_\infty}$$

Appendix B

Derivation and properties of least squares matrix

The least squares method has its origin in data fitting: so for a set of n points, x_i, $i = 1, \ldots, n$, with data values ϕ_i at each, we wish to derive a function f that best approximates the data values at these points. For a linear, polynomial fit in one dimension

$$f(x) = a + bx$$

where a and b are the coefficients to be determined. The function is computed from the minimum of the following functional

$$R^2 = \sum_i^n (\phi_i - f(x_i))^2 = \sum_i^n (\phi_i - (a + bx_i))^2$$

with respect to the coefficients a and b. The minima are obtained from

$$\begin{aligned} \frac{\partial R^2}{\partial a} &= -2 \sum_i^n (\phi_i - (a + bx_i)) = 0 \\ \frac{\partial R^2}{\partial b} &= -2 \sum_i^n (\phi_i - (a + bx_i)) x_i = 0 \end{aligned}$$

We can rearrange this system to obtain

$$\begin{aligned} na + b \sum_i^n x_i &= \sum_i^n \phi_i \\ a \sum_i x_i + b \sum_i^n x_i^2 &= \sum_i^n x_i \phi_i \end{aligned}$$

or in matrix form

$$\begin{pmatrix} n & \sum_i^n x_i \\ \sum_i^n x_i & \sum_i^n x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_i^n \phi_i \\ \sum_i^n x_i \phi_i \end{pmatrix} \tag{B.1}$$

More generally, and using the notation of Chapter 2, we wish to obtain a function $\hat{\phi}$ to approximate the data values at each point, of the form

$$\hat{\phi}(\mathbf{x}) = \sum_{k=1}^{m} p(\mathbf{x})_k \alpha_k = \mathbf{p}^T \boldsymbol{\alpha} \tag{B.2}$$

where $\boldsymbol{\alpha}$ is the local vector of coefficients that must be determined, and $\mathbf{p}$ is the vector of base monomials; both of these vectors are of dimension m, which is the order of the desired polynomial. The function that best approximates the exact values at each of the data sites will be that for which the residual

$$\mathbf{r} = \boldsymbol{\phi} - \hat{\boldsymbol{\phi}}$$

is a minimum, where $\mathbf{r}$ is a vector of dimension n. Using Eq. (2.8), the residual vector is written as

$$\mathbf{r} = \boldsymbol{\phi} - X\boldsymbol{\alpha}$$

where X is formed from the vectors of base monomials

$$X = \begin{pmatrix} p_1(x_1) & \dots & p_1(x_m) \\ \vdots & \ddots & \vdots \\ p_n(x_1) & \dots & p_n(x_m) \end{pmatrix} \in \mathbb{R}^{n \times m}$$

So for a quadratic basis in two-dimensions $m = 6$ and

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)^T$$

$$\mathbf{p} = \left(1, x, y, xy, x^2, y^2\right)^T$$

$$X = \begin{pmatrix} 1 & x_1 & y_1 & x_1 y_1 & x_1^2 & y_1^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & x_n y_n & x_n^2 & y_n^2 \end{pmatrix}$$

We use the $L2$ norm in which to measure $\mathbf{r}$ as, unlike the $L1$ norm, it is smoothly differentiable with respect to $\mathbf{r}$.

$$\begin{aligned} \min_\alpha ||\boldsymbol{\phi} - X\boldsymbol{\alpha}||^2 &= \min_\alpha (\boldsymbol{\phi} - X\boldsymbol{\alpha})^T (\boldsymbol{\phi} - X\boldsymbol{\alpha}) \\ &= \min_\alpha (\boldsymbol{\phi}^T\boldsymbol{\phi} - \boldsymbol{\phi}^T(X\boldsymbol{\alpha}) - (X\boldsymbol{\alpha})^T\boldsymbol{\phi} + (X\boldsymbol{\alpha})^T(X\boldsymbol{\alpha})) \\ &= \min_\alpha (\boldsymbol{\phi}^T\boldsymbol{\phi} - \boldsymbol{\phi}^T X\boldsymbol{\alpha} - \boldsymbol{\alpha}^T X^T\boldsymbol{\phi} + \boldsymbol{\alpha}^T X^T X\boldsymbol{\alpha}) \end{aligned}$$

Now, using vector derivatives, the minimum is computed by finding the derivative with respect to $\boldsymbol{\alpha}$ of this expression, and setting it to zero.

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\alpha}}\left(\boldsymbol{\phi}^T\boldsymbol{\phi} - \boldsymbol{\phi}^T X\boldsymbol{\alpha} - \boldsymbol{\alpha}^T X^T\boldsymbol{\phi} + \boldsymbol{\alpha}^T X^T X\boldsymbol{\alpha}\right) &= 0\\ -(\boldsymbol{\phi}^T X)^T - X^T\boldsymbol{\phi} + ((X^T X)^T + X^T X)\boldsymbol{\alpha} &= 0\\ -2X^T\boldsymbol{\phi} + 2X^T X\boldsymbol{\alpha} &= 0\end{aligned}$$

This leads to the normal equations, the same as in Eq. (B.1); and their solution leads to the coefficients to be found in Eq. (B.2).

$$X^T X\boldsymbol{\alpha} = X^T\boldsymbol{\phi}, \tag{B.3}$$

The matrix $X^T X$ is called the least squares matrix. The normal equations are equivalent to saying that the residual is orthogonal to the column space of the least squares matrix. As a result, the normal equations always have a solution even when X is not of full column rank, moreover any solution of the normal equations solves the least squares problem.

It is clear that the least squares matrix is symmetric from its form $X^T X$. For the example of a quadratic basis in two-dimensions, it can be written explicitly as

$$X^T X = \begin{pmatrix} \sum_i^n 1 & \sum_i^n x_i & \sum_i^n y_i & \sum_i^n x_i y_i & \sum_i^n x_i^2 & \sum_i^n y_i^2 \\ \sum_i^n x_i & \sum_i^n x_i^2 & \sum_i^n x_i y_i & \sum_i^n x_i^2 y_i & \sum_i^n x_i^3 & \sum_i^n x_i y_i^2 \\ \sum_i^n y_i & \sum_i^n x_i y_i & \sum_i^n y_i^2 & \sum_i^n x_i y_i^2 & \sum_i^n x_i^2 y_i & \sum_i^n y_i^3 \\ \sum_i^n x_i y_i & \sum_i^n x_i^2 y_i & \sum_i^n x_i y_i^2 & \sum_i^n x_i^2 y_i^2 & \sum_i^n x_i^3 y_i & \sum_i^n x_i y_i^3 \\ \sum_i^n x_i^2 & \sum_i^n x_i^3 & \sum_i^n x_i^2 y_i & \sum_i^n x_i^3 y_i & \sum_i^n x_i^4 & \sum_i^n x_i^2 y_i^2 \\ \sum_i^n y_i^2 & \sum_i^n x_i y_i^2 & \sum_i^n y_i^3 & \sum_i^n x_i y_i^3 & \sum_i^n x_i^2 y_i^2 & \sum_i^n y_i^4 \end{pmatrix}$$

The large powers of x and y can cause some ill-conditioning in the matrices; thus, the conditioning increases (gets worse) as m is increased.

The least squares matrix also has the property of being positive-definite; this means that for all non-zero column vectors $\mathbf{z}$ of n real numbers, the following property for a symmetric matrix A holds

$$\mathbf{z}^T A\mathbf{z} > 0$$

This is relatively simple to prove, and is as follows

$$\begin{aligned}\mathbf{z}^T A\mathbf{z} &= \mathbf{z}^T(X^T X)\mathbf{z}\\ &= (X\mathbf{z})^T X\mathbf{z}\\ &= |X\mathbf{z}|_2^2 > 0\end{aligned}$$

as this is the $L2$ norm, which is always positive.

Appendix C

Three-dimensional computational geometry algorithms

C.1 Ray-triangle intersection

1. Determine where (if at all) the infinitely long ray between the points a and b intersects the plane that contains the triangle Fig. C.1(a). The point of intersection can be found using the parametric form of the equation describing the ray between a and b, which is

$$\mathbf{l} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}) \tag{C.1}$$

where the value of t determines the intersection point. Clearly, if the value d, given by

$$d = (\mathbf{b} - \mathbf{a}).\mathbf{n}$$

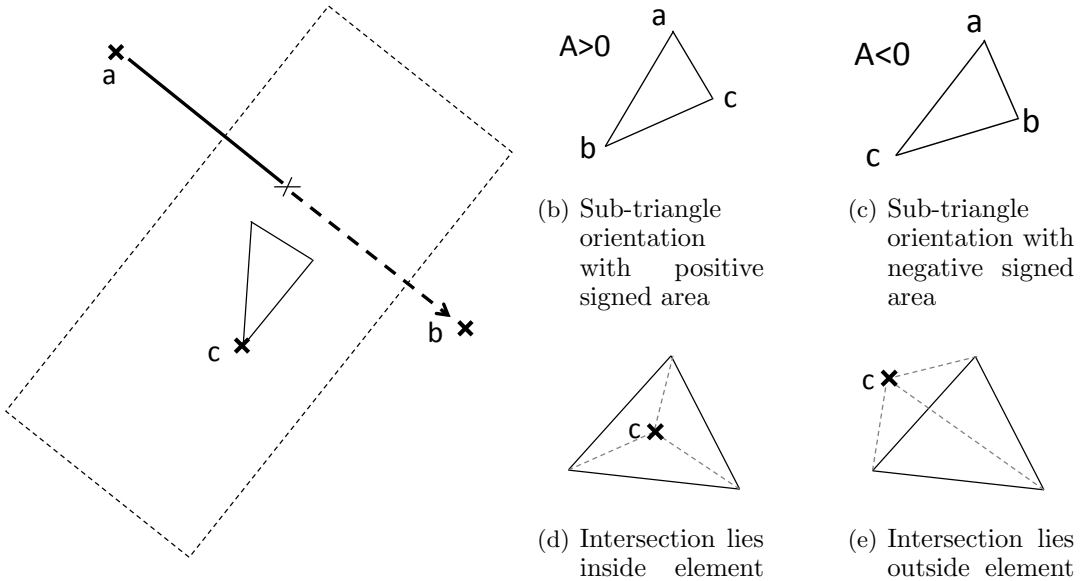
where $\mathbf{n}$ is the normal of the plane, is equal to zero then the ray is parallel to the plane and never intersects. If d is non-zero, then the infinitely long line $\mathbf{l}$ intersects the plane at some point, though it is not necessarily within the segment formed between a and b. It remains to find the value of t, which is done by first evaluating the scalar product

$$e = \mathbf{n}.(\mathbf{c} - \mathbf{a})$$

where $\mathbf{c}$ is some point on the plane (any of the coordinates of the triangle is sufficient), then t can be found from

$$t = \frac{e}{d}$$

If $0 \leq t \leq 1$ then the intersection occurs along the line segment between a and b; if $t < 0$ or $t > 1$ then the intersection is somewhere else along the line. The value of t can be inserted into Eq. (C.1) to determine the intersection point $\mathbf{x}$.



(a) Ray-plane intersection

(b) Sub-triangle orientation with positive signed area

(c) Sub-triangle orientation with negative signed area

(d) Intersection lies inside element triangle

(e) Intersection lies outside element triangle

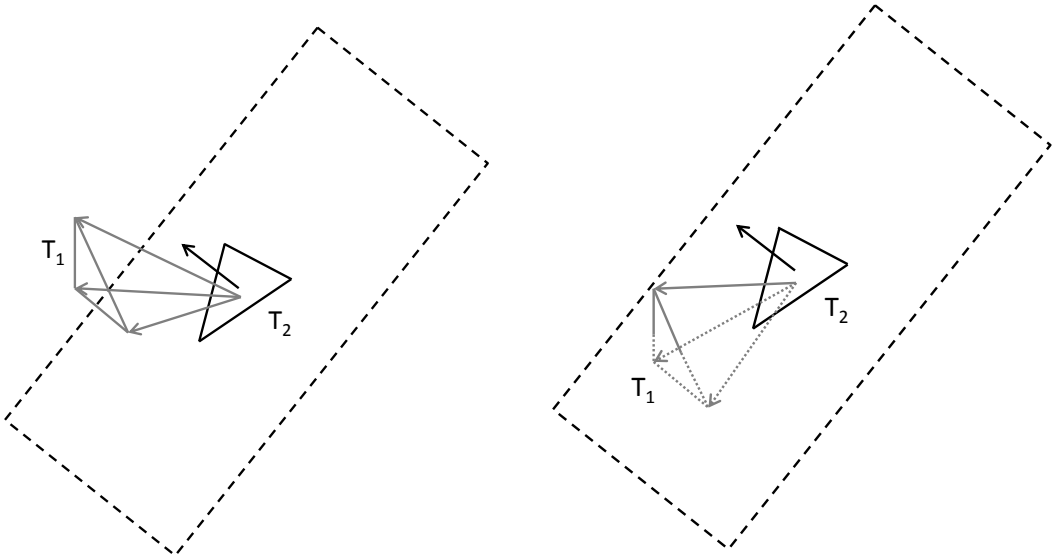
Figure C.1: Ray-triangle intersection test.

2. Change the 3D coordinate system to a 2D basis, so that the triangle coordinates and intersection point are given in xy coordinates. Using the plane as the new basis means that degeneracy is avoided. We then determine if the intersection point lies within the triangle in this new basis. This can be determined by evaluating the signed area A of the three sub-triangles, formed by the edges of the initial triangle, denoted ab, and the intersection point c, such that

$$A = \frac{1}{2}\begin{vmatrix} a_0 & a_1 & 1 \\ b_0 & b_1 & 1 \\ c_0 & c_1 & 1 \end{vmatrix}$$

The unsigned area is positive if the coordinates a, b, c are arranged in an anti-clockwise direction, Fig. C.1(b); and is negative if they are arranged in a clockwise direction, Fig. C.1(c). Consequently, if the intersection point is inside the triangle then the signed area of each of the sub-triangles will all have the same sign, Fig. C.1(d); if the intersection point is outside the triangle, then the signed area of the sub-triangles will not have the same sign, Fig. C.1(e).

If $0 \leq t \leq 1$ from Step 1 and the point lies within the triangle from Step 2, then the ray segment intersects the triangle. This test was stated in two steps for brevity; for more information see Ref. [133].



(a) Triangle lies above the plane (b) Triangle lies within the plane

Figure C.2: Testing if the triangles lie in the same plane.

C.2 Triangle-triangle intersection

Define each of the triangles to be tested against as T_1 and T_2, and the planes that contain them as π_1 and π_2 respectively.

1. First we test if the plane containing T_1 intersects T_2; this can be done using the implicit form of the plane equation containing T_1

$$\pi_1 : \mathbf{n}_1.\mathbf{x} + c = 0 \tag{C.2}$$

where $\mathbf{n}_1$ is the normal of the plane, $\mathbf{x}$ is a point coordinate, and c is a constant such that the equation is satisfied if the point $\mathbf{x}$ lies in the plane. The normal can be determined from the cross product of two vectors within the plane; these can be obtained from

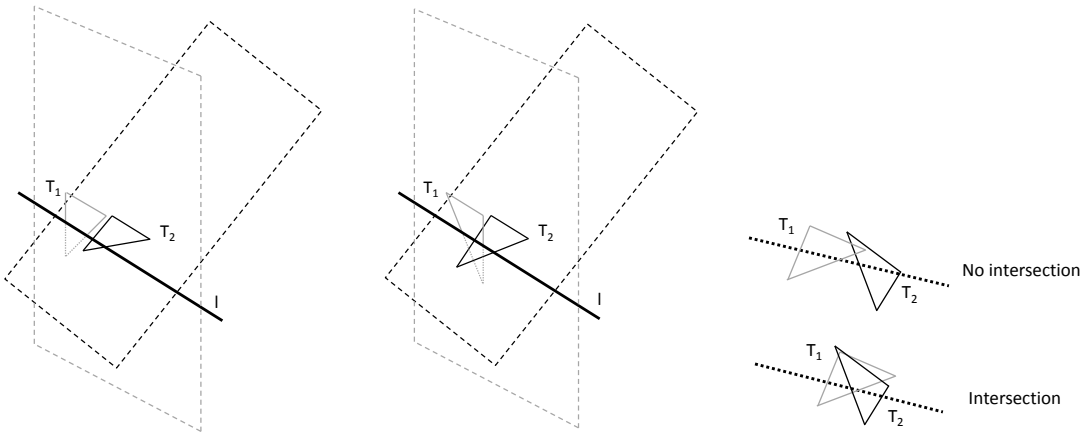
$$\mathbf{n}_1 = (\mathbf{v}_{(1)1} - \mathbf{v}_{(1)0}) \times (\mathbf{v}_{(1)2} - \mathbf{v}_{(1)0})$$

where $\mathbf{v}_{(1)0}$, $\mathbf{v}_{(1)1}$ and $\mathbf{v}_{(1)2}$ are the coordinates of the vertices of T_1. The constant c can be computed from

$$c = -\mathbf{n}_1.\mathbf{v}_{(1)0}$$

this means that if $\mathbf{x}$ lies within the plane, then Eq. (C.2) is satisfied because

$$\mathbf{n}_1.(\mathbf{x} - \mathbf{v}_{(1)0}) = 0$$



(a) The intervals of the triangles on the plane of intersection do not overlap

(b) The intervals of the triangles on the plane of intersection do overlap

(c) Projection of the line and triangles to determine if intervals overlap

Figure C.3: Tests to see if the triangles intersect using the intervals of the triangles on the line of intersection between the planes containing them.

as the vector $(\mathbf{x} - \mathbf{v}_{(1)0})$ lies orthogonal to the plane normal. If we insert the vertices of T_2 into the plane equation for π_1 then we obtain the signed distances from the vertices of T_2 to π_1

$$d_{\mathbf{v}_i} = \mathbf{n}_1.\mathbf{v}_{(2)i} + c \qquad i = 0, 1, 2 \tag{C.3}$$

If all of the signed distances have the same sign (that is $d_{\mathbf{v}_i} > 0$ for $i = 0, 1, 2$ or $d_{\mathbf{v}_i} < 0$ for $i = 0, 1, 2$) then T_2 lies completely on one side of π_1 and the overlap is rejected, Fig. C.2(a). If this is not the case, then the plane π_1 intersects T_2, Fig. C.2(b), and we must then check for an intersection of π_2, and the triangle T_1. If both of these intersections occur, then it is possible that the triangles intersect; and it also means that there must be a line in 3D that determines the intersection between the planes, and also passes through each of the triangles. This line is necessary for Step 2.

2. Step 1 ensures that there is a line $\mathbf{l}$, formed at the intersection of both planes π_1 and π_2, and that this line intersects both triangles. The intersection of the triangles form intervals on $\mathbf{l}$, and if these intervals overlap then the triangles intersect, Fig. C.3. The line $\mathbf{l}$ can be written in the form

$$\mathbf{l} = \mathbf{p} + t\mathbf{d}$$

where $\mathbf{p}$ is a point which the line passes through, $\mathbf{d}$ is a vector that points in the direction of the line, and t is a scalar which defines a point along the line. The

line must be perpendicular to both planes, and so $\mathbf{d}$ can be found simply from

$$\mathbf{d} = \mathbf{n}_1 \times \mathbf{n}_2$$

The intervals can be found by determining the intersection points between $\mathbf{l}$, T_1 and T_2: in computational geometry it is much more robust to do this in a 1D projection of $\mathbf{l}$ and the vertices v of the triangles. For this it is first necessary to choose in which direction to make the projection. Since the intervals can be translated without altering the results of the overlap test (which will be done in Step 3), it is simplest to project $\mathbf{l}$ onto the coordinate axis to which it is most closely aligned to avoid degeneracy, so

$$Pv_i = \begin{cases} v_{ix}, & \text{if } |d_x| = max(|d_x|, |d_y|, |d_z|) \\ v_{iy}, & \text{if } |d_y| = max(|d_x|, |d_y|, |d_z|) \\ v_{iz}, & \text{if } |d_z| = max(|d_x|, |d_y|, |d_z|) \end{cases}, \quad i = 0, 1, 2 \tag{C.4}$$

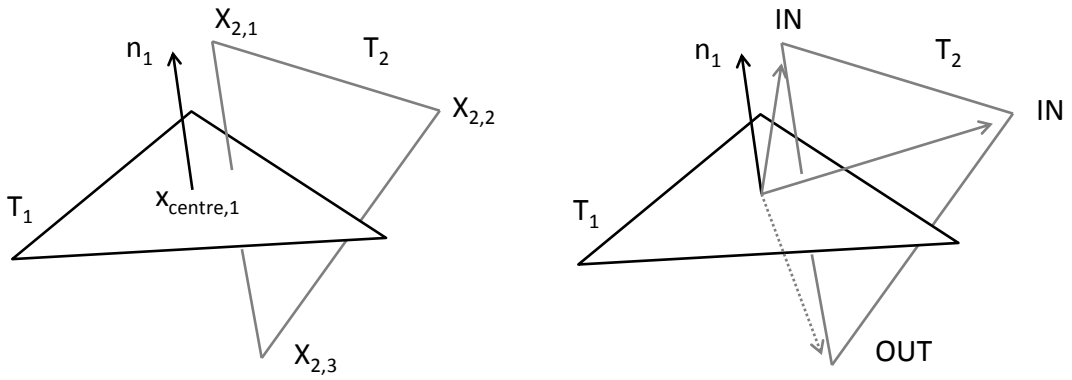
where v_{ix} means the x component of v_i and so on. Such a projection means that it is unnecessary to compute $\mathbf{p}$.

3. We now need to compute interval values t along this coordinate axis that represent the intersection of the line, and the segments formed by the two pairs of vertices of T_1 that have opposite signed distances as calculated in Step 1. So if the segment is between the vertices v_0 and v_1, then t is calculated from

$$t = Pv_0 + (Pv_1 - Pv_0)\frac{dv_0}{dv_0 - dv_1}$$

t is then calculated for the other segment. The smaller of these values is denoted t_1, while the larger is denoted t_2; the same is done for T_2, this time the smallest valus of t is denoted t_3, and the larger is t_4. If the interval values overlap, that is $t_3 > t_1$ and $t_2 > t_3$, or $t_1 > t_3$ and $t_4 > t_1$, then the triangles intersect.

For more details on this method, and for information regarding cases when the triangles are nearly coplanar, or when an edge is near coplanar to the other triangle, see Ref. [136].



(a) Labels of points when two triangles intersect (b) Flag the points of T_2 using the normal of T_1

Figure C.4: Flag points belonging to two intersecting triangles.

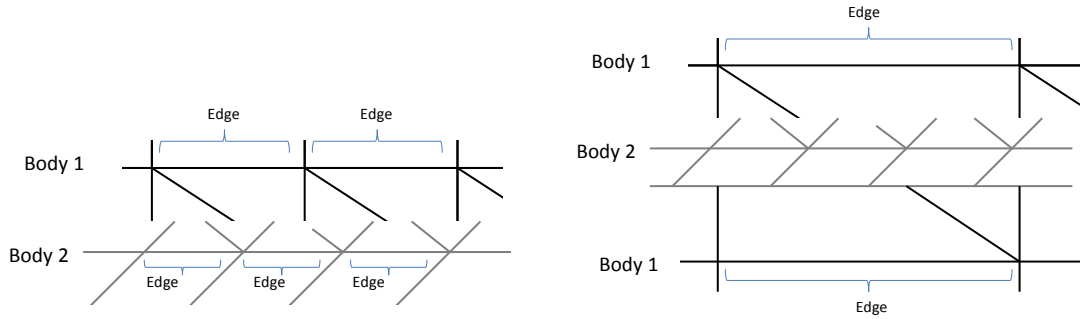
C.3 PML boundary reallocation

This section describes, in six steps for brevity, the algorithm employed by PML to reallocate the boundary elements, when the solid wall geometry of two input domains to the preprocessor intersects in some way. The elements (which have been split up into triangles to ensure robust computational geometry) that intersect were identified using the triangle-triangle intersection algorithm described in Appendix C.2; and the following algorithm creates triangles as new elements between the bodies to form the new boundary surface.

1. The first step is to determine which surface points need to be blanked as a result of the intersection: this can be done with the use of normal tests. We have the correct normals of each element due to the reference points that are part of the required input. When two triangles intersect, we can use the normal test

$$\mathbf{n}_1.(\mathbf{x}_{2,i} - \mathbf{x}_{1,centre}) \geq 0 \qquad i = 0, 1, 2 \tag{C.5}$$

where $\mathbf{n}_1$ is the normal of one triangle and $\mathbf{x}_{1,centre}$ is its centre, and $\mathbf{x}_{2,i}$ is each of the vertices of the triangle that intersects it, as is shown in Fig. C.4(a). If Eq. (C.5) is satisfied, then the point is flagged as IN, so remains within the domain; if not, then the point is flagged as OUT *if* it has not already been flagged as within the domain, Fig. C.4(b). As the elements are triangles and intersect, then it is safe to use this test without worrying about the possible concave nature of boundary walls. After this test has been performed for all of the vertices that make up the intersecting triangles, the points that remain flagged as OUT are blanked and removed from the computational domain. A flood is then performed to identify and remove the boundary points that are connected to the blanked points found. These points belong to elements for which there is no intersection, but must lie inside a solid body.



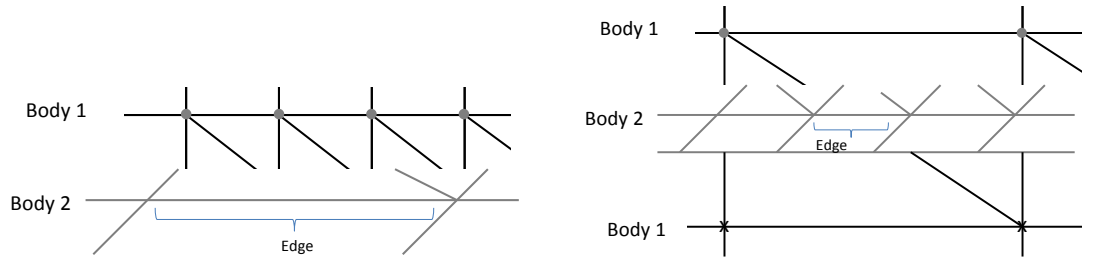
(a) Simple case in which edges with no blanked points are retained

(b) Geometric tests are needed because the diagonal edge intersects Body 2, but contains no blanked points

Figure C.5: Define the edges to be used in triangulation.

2. Next it is necessary to define the edges, from the elements of each body, that will make up the triangulation. These edges are formed within the original intersecting elements, so are not formed between the two separate bodies. The triangles that are to make up the triangulation will consist of two points from one of these edges, and a third point from the intersecting element from the other body (done in subsequent steps). To do this it is necessary to find the edges of the elements that are not cut by another element intersecting it; obviously, if an edge consists of a blanked point then it does not fit this criteria. The desired edges can be seen in Fig. C.5(a) for a simple case. For geometries with sharp edges or corners, however, an intersection may occur with an element, and no points of the elements are blanked, such is the case for the highlighted elements making up Body 1 in Fig. C.5(b). Because of this, it is necessary to explicitly check all of the complete edges belonging to the intersecting elements to determine if these edges intersect other elements at all. The ray-triangle algorithm of Appendix C.1 is used to do this, where the ray consists of the edge to be checked. If an edge intersects an element, then it is deemed incomplete and discarded.

3. Next we create a list of points, per edge that was found in Step 2, that can form the third point of a triangle. The list consists of non-blanked points that form the elements that intersect the element of which the edge is a part, as seen in Fig. C.6(a). Some of these points may not be acceptable for a particular edge to form a triangle with; this may be the case at corners or sharp edges of the geometry, or where a body made up of coarse elements intersects a much smaller component. For example, Fig. C.6(b) shows a sharp body completely intersecting an element from another body; it should be noted that there is another (unseen) surface, belonging to Body 2, located below the surface containing the edge: this forms the underside of the sharp geometry of Body 2. It is clear that not all of the points from the element can form a triangle with the highlighted edge



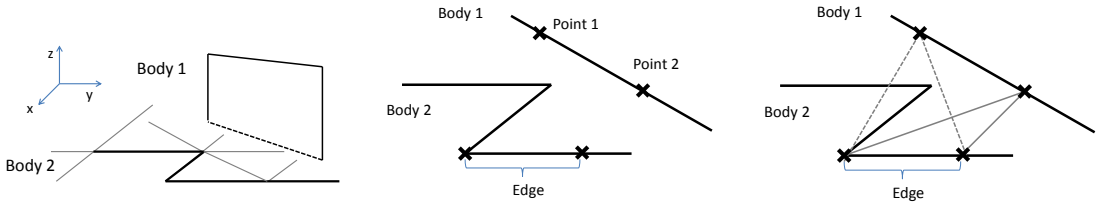
(a) Candidate points from Body 1, for the triangle to be created using the edge from Body 2

(b) For the edge shown, points denoted with dots can form triangle, those with crossses cannot

Figure C.6: Candidate points per edge to be used in triangulation.

to form a continuation of the geometry. In this case, the points indicated by grey circles are physically possible, whilst those indicated by crosses must be removed from the candidate list for the edge. Unfortunately normal tests are not sufficient for eliminating points that do not conform to the resulting (or current) boundary contours. Normal tests were sufficient in Step 1 in identifying blanked points because if *ever* a point is identified as lying within the domain it is flagged as within, regardless of the number of times it fails the normal test; but it is entirely possible in three dimensions for reallocated boundaries to point in a direction more than 90° from the normal of the element containing the edge. It is preferable, instead, to use ray intersection algorithms for each of the points in the list. A ray is formed between one of the points forming the edge and the point in the list; if this ray intersects any of the neighbouring elements then the point is removed from the list; if not, then a ray is formed between the point in the list and the other point that forms the edge, and the test is performed again. For efficiency, the neighbouring elements can be found using the 6D search trees that were created to identify the boundary overlap; and the seach region is defined by a bounding box formed around the ray. The remaining points from the list, for which there is no ray intersection, form the next list of candidate points for the subsequent steps.

4. Although the points in the list obtained in Step 3 respect the geometry of the problem, some of the points in the edge list cannot be used as they will result in an overlap of elements on the wall. These overlaps will cause holes to appear in the geometry, which will mean that the rest of the preprocessor will fail because the geometry is not closed. Such a case can be seen in Fig. C.7(a), which shows the intersection of two walls; for clarity a 2D projection, looking down on the xy plane, is shown in Figs. C.7(b) and C.7(c). For the highlighted edge, a triangle formed with Point 1 will overlap the element that is not altered by an intersection, Fig. C.7(c); the triangle formed by Point 2 is more preferable. For each point



(a) View of two bodies intersecting, only one element from Body 1 is shown

(b) Projection of Fig. C.7(a) onto the xy plane

(c) Possible triangles formed from the edge highlighted

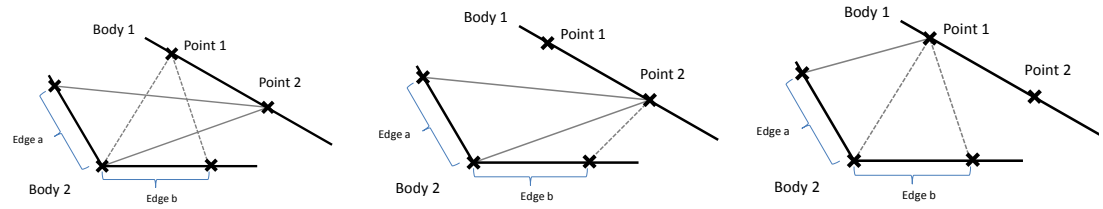
Figure C.7: Remove non-conforming boundary points from candidate list.

in the candidate list we form the triangle that would be created with the edge, then using a 2D projection we test for overlap with the neighbouring elements [1] *provided* that the angle between the normals of the triangle and element tested against is less than 90° (so the scalar product of both normals is positive). This prevents testing for overlap between two elements that point in different directions (i.e. are on opposite sides of a geometry) and, hence, there is no danger of overlap. In some (rare) cases where there is very complicated geometry, such as double folds or extreme curvature of the wall, all of the candidate points will fail this test. As a result, the points identified using the test in this step are not completely removed from the list of candidates. Instead, they form a second list of candidate points, which are less preferable; these are only to be used if there are no points in the list that pass this test, or, if there are, they can be used as a last resort for the triangulation.

5. We now have, for each edge, two lists of points: one of which contains more preferable points compared to the other list. Now we perform the triangulation, for which we select a point for each edge from the intersecting body. This process can be split into two further steps, because the triangulation is performed for all of the edges belonging to one of the intersecting bodies first, then all of the edges belonging to the other body second.

   (a) The triangles formed are based on a Delaunay triangulation, which means that the triangles are less "stretched", and are as close to being equilateral as possible. Thus, each possible triangle in the first list of points is formed, and each of the interior angles is calculated. The smallest of these angles for each triangle is stored; and the list of points is rearranged so that the point which forms the triangle with the largest of these angles is first: this first point forms the triangle. It may be possible for some of these triangles, from different edges, to overlap, as in Fig. C.8(a); so each of the triangles

[1] The same elements used for the tests in Step 3 are sufficient, though only use those elements that are not blanked and are unaffected by an intersection.



(a) Two triangles from Body 2 overlapping

(b) If the triangle from Edge a is made first, the triangle from Edge b is reallocated

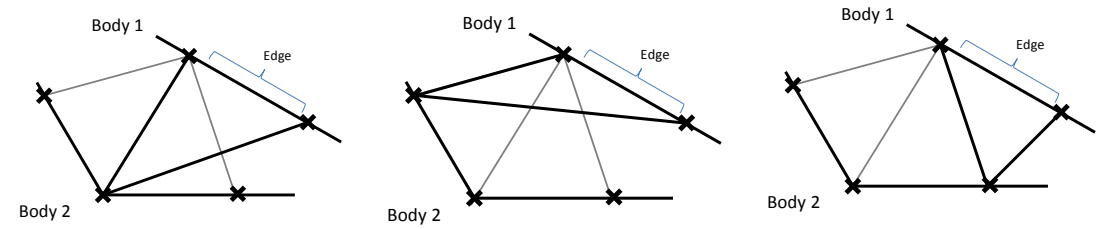
(c) If the triangle from Edge b is made first, the triangle from Edge a is reallocated

Figure C.8: Overlap of triangles belonging to same body.

that have already been created are checked against the new triangle that has just been created. If the third point of the triangle to be checked against is in the list of candidate points for this triangle then the same projection test used in Step 4 is employed, else it is ignored. If an intersection occurs, then the triangle being created uses the third point of the intersecting triangle as its third point; this process is seen in Figs. C.8(b) and C.8(c). This step will result in triangles formed between the two bodies using the complete edges from the first body.

(b) Next the triangles are formed from the edges belonging to the second body. This step first follows the same procedure as in Step 5a: the reordering of points in the list is based on a Delaunay triangulation, and the first point is used in the triangle. This time though, the triangles that the new triangle is checked against will clearly only belong to the second body (as the third point of the triangles formed with the first body in Step 5a cannot belong to the list of candidate points for any edge belonging to this body). As a result, we need some way of checking for overlap against the triangles created in Step 5a; so, if either of the first two points of the triangles formed in Step 5a are points within the candidate list for the new triangle, it is checked using the same overlap test as in Step 4. If there is an intersection, then the next point in the candidate list is used, as is done in Fig. C.9. This continues until the correct point is found.

6. The above steps will give a new surface between the two bodies, made up of triangles, the majority of which are non-overlapping, resulting in no holes in the surface. There may be one or two erroneous elements, which occur when there is a double fold in the geometry, Fig. C.10(a). The tests performed in Steps 4 and 5 are only performed when the normal of the triangle and the neighbouring elements have an angle between them of less than 90°. For the case in Fig. C.10(a) the highlighted element has *no* neighbouring elements that satisfy this (the normals of each element is denoted by an arrow). As a result, no tests are performed and



(a) The triangle formed by the highlighted edge of Body 1 overlaps a triangle from Body 2

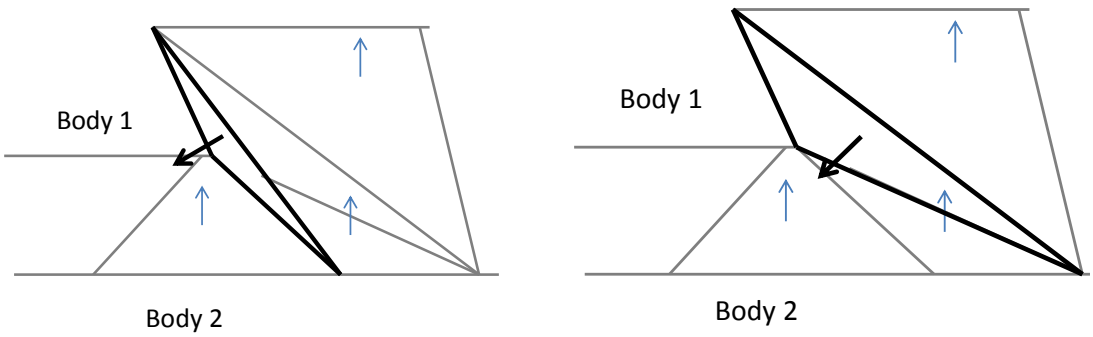
(b) The next candidate point is used for the triangle but there is still an overlap

(c) The last candidate point is used for which there is no overlap

Figure C.9: Overlap of triangles belonging to opposite bodies.

the wrong point is chosen in the triangulation. For such triangles it is necessary to identify the triangles that lie either side of it in the same body; these are found easily because their initial edges share a common point. If one of these neighbouring triangles has the same third point as the highlighted triangle then it is left alone, else the next candidate point is used until this is the case; this closes the geometry as seen in Fig. C.10(b).

The above algorithm is very detailed, but it does ensure that the resulting boundaries are robustly rewritten, with no holes in the surface. Due to the procedure of Step 5, the order that the geometries are input into the preprocessor does affect the resultant surface discretisation. This is not ideal, and future work should eliminate this property.



(a) The highlighted triangle is created incorrectly, creating a hole in the surface

(b) The erroneous element is fixed by choosing the next candidate point

Figure C.10: Element belongs to a unique plane.